MACOBJ - A Class-Based Tool for Numerical Problem Solving

Alfio Ricardo Martini Marcio Serolli Pinho
Tiaraju Asmuz Diverio, Msc

Instituto de Informatica e PGCC da UFRGS - RS - Brazil
Fax : (051)3365576
E-Mail : MARTINI@INF.UFRGS.BR

Abstract

The purpose of this paper is to present a tool for numerical problem solving (MACOBJ), which is based on a command language interaction style user-interface. The user-interface provides the the user with a very simple and consistent conceptual model, which is based on the object-oriented paradigm. Firstly,we introduce the notion of object-oriented user-interfaces. After, we present the classes of mathematical objects offered to the user, and an overview of the system's functionality and its symbolic capabilities.

Key words: object-oriented design, command languages, numerical calculus, conceptual model.

1 Introduction

MACOBJ represents a natural evolution from a previous design [5] which was also based on a command language interaction style.

After prototyping a series of test designs, we concluded that a very simple and coherent user's model will have to be developed so as to overcome the known problems related to command languages based user-interfaces [7].

On our last design we proposed the following requirements for the software tool :

- It should cover a *broad array* of computational methods;

- It should be *objective* (i.e., with a few steps the user should accomplish what it wanted) and *powerful* enough to be used by domain experts (e.g., teachers and researchers of the related field);
- It should present the user with a very well defined and *simple conceptual model*, so as to enable him (or her) to apply always the same semantic strategies throughout his (or her) application tasks;

The object-oriented design methodology provided us with useful concepts to create a very consistent and simple user's model, as well as to refine our command language grammar, by applying orthogonalization (factorization) [3] on some language parameters from our previous language description.

Throughout the following, we discuss basic object-oriented concepts related to user-interfaces and MACOBJ's basic functionality. We assume that the reader is familiar with the basic object-oriented terminology, and its benefits to the computer field as well ([4], [BOO91]).

2 Object-Oriented User-Interfaces

An object-oriented system can be understood as a collection of objects. Each object implements a little piece of the total system or application behaviour.

When building an object-oriented user-interface one of the first things to deal with is the global system's object organization. In order to design and implement an user-interface component , the designer needs to specify what kind of communication can occur between the objects. The possible messages which can be sent or received by an object must be clearly defined in its class interface (message protocol).

An object-based user interface enables one to create objects which are typeless (not belonging to any class). A class-based user interface permits the creation of typed objects, and in an object-oriented user-interface it is possible to create typed objects that, possibly, inherit some properties from some other classes [8] (see Fig. 1).

MACOBJ's user-interface is class-based, because it does not yet provide any mechanism for the user to express inheritance relationships between the objects created during the application task.

3 The Conceptual Model

The user's conceptual model is represented by the information handled by the user and the processes applied to this information [6]. The model enables him (or her) to

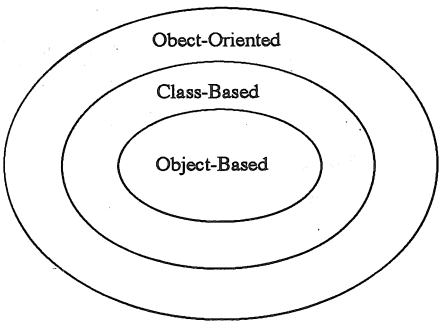

Figure 1: Object-Orientation and User-Interfaces

develop, with or without computer knowledge, a broad comprehension about what the program is doing. By understanding the conceptual model, he can anticipate the effect of his actions and develop strategies to correctly operate the program.

In the MACOBJ's conceptual model, the user creates an object by issuing a command that simultaneously define the object's name and its class. Therefore, when wanting to perform some computation on the object, the user needs just to edit the appropriate command (with its necessary parameters, if any), and it will automatically apply to the *current selected object* (*CSO*), i.e., the one that was last created. The CSO paradigm (as defined in [3]) was obtained by factoring the object parameter from our previous grammar definition. The CSO paradigm also enables that all the common operations that are applicable to objects of any class, be issued with the same syntax. The appropriate method is determined by the *current selected class*. This approach relieves the name space congestion problem. That it is to say, he , in a given state of its application task, needs just to concentrate on a smaller sets of commands, rather than worrying with all the ones available in the tool's user-interface.

4 System's Functionality.

MACOBJ's user-interface presents the user with four pre-defined mathematical classes (*Mathematical Expressions*, *Ordered-Pairs*, *Variables* and *Differential Equations*) upon which he can create objects and realize a series of computations. The E-R [2] diagram, in Fig. 2, present these classes and how they are related to each other.

The interpretation of the E-R diagram is straightforward, and it says, for instance, that an object of the class *Mathematical Expressions* can create up to n (n = 1, 2, 3,...) objects of the class *Ordered-Pairs*. Also, an object of the class *Mathematical Expressions* can use resources from up to n (here, n is implementation defined) objects

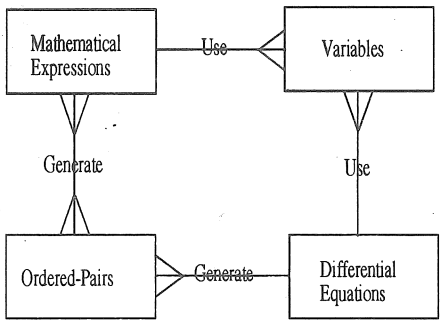

Figure 2: E-R diagram for MACOBJ's classes

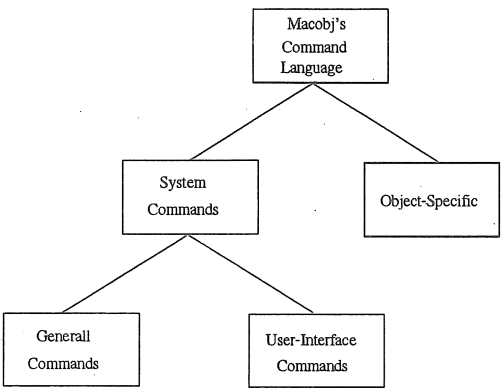

Figure 3: MACOBJ's command language hierarchy

of the class *Variables*. These relationships are supported by a series of methods offered by each class interface.

Some commands, so-called *system commands*, can be issued at any time. Other commands, *object specific commands*, can only be issued once an object has been selected. So, the range of object-specific commands is determined by the type (class) of the *CSO*, and will vary from class to class. The system commands are subdivided in *generall commands*, i.e, its behavior depends upon the current selected class, and *user-interface commands*, which are commands for object visualization and context sensitive help, basically (Fig. 3).

A major advantage of the class based style of command invocation is that it permits user's to concentrate on what it is they want to accomplish, and not how it is that this task is to be performed. Individual objects themselves are responsible for knowing the how, leaving the user to work at a much higher level.

In order to present a broad array of computational methods, MACOBJ's user-interface enables the user to perform computations of zero finding, polynomial fitting, numerical integration, derivative evaluation, and ordinary differential equations solving.

5 Conclusions

In this paper we have described a class-based tool for numerical problem solving, MACOBJ.

We used object-oriented concepts to create an suitable abstraction to develop MACOBJ's conceptual model. The conceptual model is based on the CSO paradigm, and it is enabled by applying a process called orthogonalization on the object parameter of the command language grammar. This paradigm enables the user to see the application as a consistent model, letting him develop homogeneous strategies to operate all the objects in the application regardless of its class (type). Also, the command language interaction style, together with a broad array of computational methods, represents a powerful tool for the domain expert user to quickly solve the common problems related to numerical calculus.

References

[1] BOOCH, G. *Object-Oriented Design with Applications.* The Benjamin/ Cummings Publishing Company, Inc, California, 1991.

[2] CHEN, P. P.,The Entity-Relationship Model - Toward a Unified View of Data, *ACM Transactions on Database Systems* , 1(1), pp. 9-36 (March 1976).

[3] FOLEY, J. Model and Tools for the Designers of User-Computer Interfaces. In : *Theoretical Foundations of Computer Graphics and Cad* , Italy, July 1987.

[4] MEYER, B. *Object-Oriented Software Construction* , Prentice-Hall, New-York, 1988.

[5] MARTINI, A.R. and DIVERIO, T.A. A Descricao de uma Interface Grafica e Interativa para Matematica Computacional. CONFERENCIA LATINO-AMERICANA DE INFORMATICA, XVII, 1991, 8-12 julho, Caracas,*Annales*, CLEI 1991, pp. 949-965.

[6] NEWMAN, M.W. and SPROULL, F.R. *Principles of Interactive Computer Graphics* . MacGraw-Hill, New York, 1979

[7] SHNEIDERMANN, B. *Designing the User-Interface : Strategies for Effective Human-Computer Interaction* . Addison-Wesley, Reading, 1987.

[8] WEGNER, P. Learning the Language, *Byte*, pp. 245-253, March 1989.