The Implementation of ObjectMath – a High-Level Programming Environment for Scientific Computing

Lars Viklund, Johan Herber and Peter Fritzson

Programming Environments Laboratory
Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Abstract. We present the design and implementation of ObjectMath, a language and environment for high-level equation-based modeling and analysis in scientific computing. The ObjectMath language integrates object-oriented modeling with mathematical language features that make it possible to express mathematics in a natural and consistent way. The implemented programming environment includes a graphical browser for visualizing and editing inheritance hierarchies, an application oriented editor for editing ObjectMath equations and formulae, a computer algebra system for doing symbolic computations, support for generation of numerical code from equations, and routines for graphical presentation. This programming environment has been successfully used in modeling and analyzing two different problems from the application domain of machine element analysis in an industrial environment.

1 Introduction

The programming development process in scientific computing has not changed very much during the past 30 years. Most scientific software is still developed the traditional way [3]. Theory development is usually done manually, using only pen and paper. In order to perform numerical calculations, the mathematical model must be implemented in some programming language, in most cases FORTRAN. Often more than half the time and effort in a project is spent writing and debugging FORTRAN programs. The process is highly iterative, as feedback from numerical computations and physical experiments can affect both the underlying mathematical model and the numerical implementation. This iteration cycle is very time consuming and has a tendency to introduce errors, see Fig. 1.

In order to improve this situation we have designed and implemented an object-oriented programming environment and modeling language called ObjectMath. This environment supports high-level equation-based modeling and analysis in scientific computing. It is currently being used for applications in advanced mechanical analysis, but it is intended to be applicable to other areas as well. The implemented programming environment includes a graphical browser for visualizing and editing inheritance hierarchies, an application oriented editor for editing ObjectMath equations and formulae, a computer algebra system for doing symbolic computations, support for generation of numerical code from equations and for combined symbolic/numerical computations, as well as routines for graphic presentation. This paper describes the ObjectMath environment and its implementation.

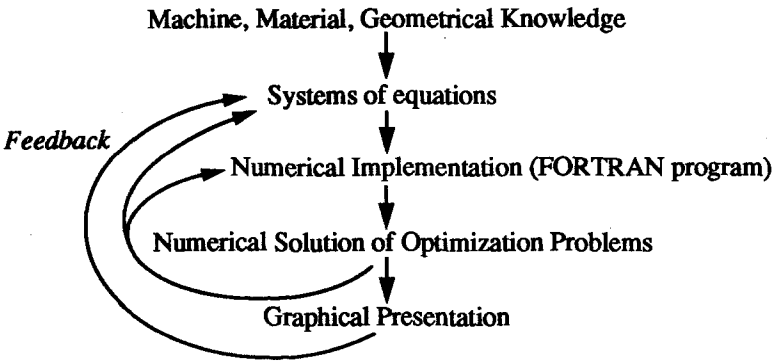

Fig. 1. The iterative process of modeling in traditional mechanical analysis

2 The ObjectMath Language

ObjectMath [8] is a hybrid modeling language, combining object-oriented constructs with a mathematical language. This combination makes ObjectMath a suitable language for implementing complex mathematical models. Formulae and equations can be written with a notation that closely resembles conventional mathematics, while the use of object-oriented modeling makes it possible to structure the model in a natural way.

We have chosen to use an existing computer algebra language, Mathematica [11], as a base for ObjectMath. The relationship between Mathematica and ObjectMath can be compared to that between C and C++. The C++ programming language is basically the C language augmented with classes and other object-oriented language constructs. In a similar fashion, the ObjectMath language can be viewed as an object-oriented version of the Mathematica language.

Ordinary Mathematica packages can be imported into an ObjectMath model. Such packages exist for a variety of application areas such as trigonometry, vector analysis, statistics and Laplace transforms. It is also possible to call external functions written in other languages. The current implementation of the programming environment supports external C++ functions, but in principle is it possible to use any language.

3 The ObjectMath Programming Environment

In this section we give an overview of the basic features of the ObjectMath programming environment. The implementation is described in the next section. Currently, the programming environment supports:

- Graphic browsing and editing of inheritance hierarchies
- Textual editing of ObjectMath code
- Interactive symbolic computation
- Automatic code generation from simple ObjectMath equations
- Mixing ObjectMath and C++ for combined symbolic/numerical computations
- Graphic presentation

The graphical browser is used for viewing and editing ObjectMath inheritance hierarchies. It is integrated with the Gnu Emacs editor for editing of equations and formulae. The Mathematica computer algebra system is also integrated within the environment. ObjectMath code is translated into pure Mathematica code by the ObjectMath translator. Algebraic simplification of equations can be done interactively in Mathematica. Figure 2 shows the screen during a typical session.

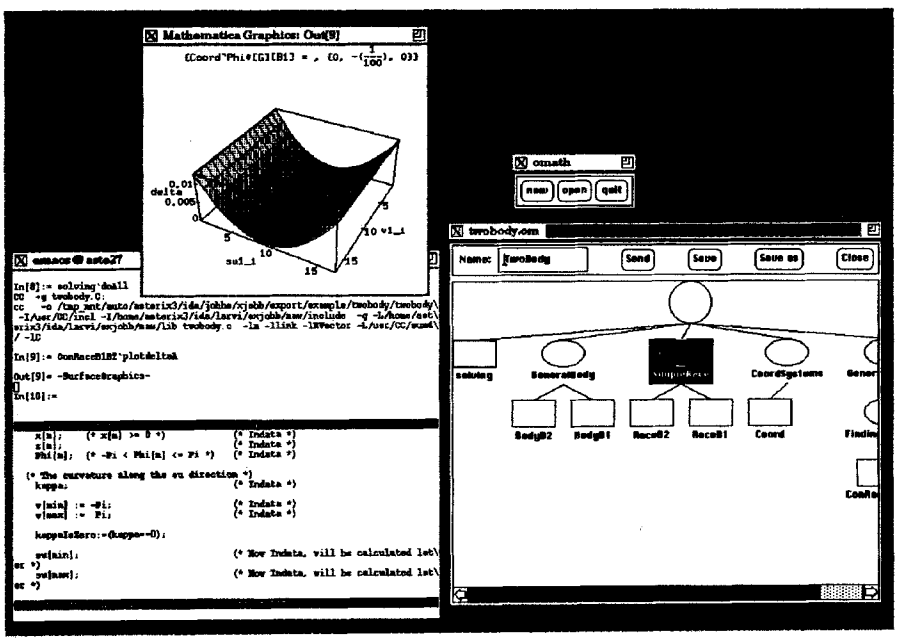

Fig. 2. The ObjectMath programming environment in use

Analyzing a mathematical model expressed in ObjectMath involves performing numerical computations. The Mathematica system can be used for some of these calculations. However, Mathematica code is interpreted and cannot be executed as efficiently as programs written in compiled languages such as C, C++ or FORTRAN. This might be a serious drawback, particularly when doing mostly numerical computations in realistic applications. It is also desirable to take advantage of the large number of existing, highly optimized, special purpose numerical routines.

The ObjectMath environment provides the possibility to generate C++ code and to mix ObjectMath and C++, thus enabling us to take advantage of symbolic computation while still being able to write time-critical functions in a language that can be compiled into efficient code. Numerical routines can either be called from within the ObjectMath environment, via an implemented message-passing protocol, or be used independently of the environment as a computation kernel, for example together with a graphical front end.

A library with general classes is also available. This includes classes for modeling simple bodies (spheres, cylinders, rings, etc.), coordinate systems and contacts between bodies. The classes for modeling bodies implement methods which generate three-dimensional plots of the bodies from the surface descriptions. Graphical support helps the user to visually verify the formulae and equations which specify geometric properties.

4 Implementation of the Programming Environment

The ObjectMath programming environment has been implemented in C++, Scheme, Gnu Emacs Lisp and Mathematica. It currently runs on Sun workstations under the X window system. The main parts are:

- The graphical browser, which allows editing of class hierarchies.
- The Gnu Emacs editor, which is used for editing of ObjectMath equations and formulae.
- The ObjectMath translator, which translates ObjectMath programs into Mathematica code.
- The Mathematica system, which runs as a subprocess to Emacs.

Gnu Emacs communicates with the ObjectMath translator and the Mathematica process via UNIX pipes, while communication with the browser is done through sockets. The passing of code from the translator to the Mathematica process utilizes a temporary file. Figure 3 shows the internal structure of the system.

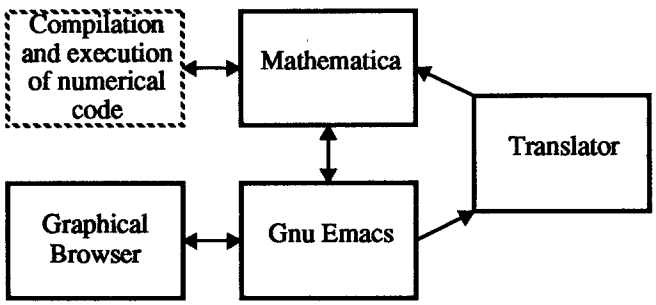

Fig. 3. Structure of the implementation

4.1 The ObjectMath Browser

The ObjectMath browser allows the user to view and edit ObjectMath inheritance hierarchies. Properties such as parameters to classes are also edited in the browser, while the equations and formulae in the classes are edited as text in Gnu Emacs. When the user selects a class or instance to be edited, an Emacs text buffer with the body of the declaration is created and displayed. The browser also has a number of command buttons, for instance one for translating the current model and sending it to Mathematica.

The browser is implemented in C++ using the ET++ class library [10]. ET++ includes classes for building user interfaces and general classes such as different kinds of collections. The object-oriented design of ET++ makes the library very flexible. Advanced classes can be utilized even if they do not fit exactly into the application being implemented by defining a new class inheriting from a suitable existing class and redefining a few methods.

4.2 Customizing the Gnu Emacs Editor

Most of the features in the Gnu Emacs editor are written in a Lisp dialect called Gnu Emacs Lisp [6]. Gnu Emacs can easily be extended by writing new Lisp code and installing it as an

extension to the editor. Emacs Lisp is a full programming language, with additional features for handling editor specific functions such as text buffers. In the ObjectMath environment, Gnu Emacs has been extended with a special ObjectMath mode. A separate Emacs buffer is used for each class or instance declaration. Switching between different object buffers is done by selecting their icons in the browser window using the mouse.

4.3 Communication between Gnu Emacs and Other Subsystems

The communication between the browser and Emacs goes through a pair of sockets. When the browser is started, it forks a process in which Emacs is executed. After this, the browser creates a socket and listens to it. Emacs starts a communications subprocess which creates its own socket and opens a connection to the browser. Whenever the user issues a command in the browser that affects data in Emacs one or more messages are passed to Emacs in order to keep the data structures up to date, save modified files, etc.

4.4 Translating ObjectMath Programs

ObjectMath programs are translated into Mathematica packages by a series of program transformations. The Mathematica *context* facility is used to implement objects. A Mathematica context provides a separate name space, similar to a block in Algol-like languages. Packages generated from ObjectMath models consists of a number of context declarations, one for each instance. Retranslation in the ObjectMath translator is incremental with the granularity of an object. Therefore, new Mathematica code will be produced very fast if only a single instance is changed.

A first version of the ObjectMath translator was implemented in Gnu Emacs Lisp. Unfortunately, this implementation turned out to be too inefficient for practical use, forcing us to re-implement it in Scheme. The Scheme program was compiled with Bartlett's Scheme→C compiler [1] and runs about 50 times faster than the original Emacs Lisp implementation.

4.5 Generating Numerical Code from ObjectMath Equations

The ObjectMath environment supports automatic generation of numerical code for solving sets of non-linear equations. Generated code is linked to numerical FORTRAN routines which perform the actual solving. Currently, we use the MINPACK [5] routines HYBRD and HYBRJ as solvers. One of the input parameters to these is a routine which calculates the values of the functions. This routine is generated from a C++ code template, using a translator which generates three-address statements expressed in C++ from ObjectMath expressions. The ObjectMath→C++ translator does common subexpression elimination, using the fact that functions such as *sin*, *cos* etc. do not have side effects.

4.6 Mixing ObjectMath and C++ Code

The ObjectMath environment allows C++ functions to be used as ObjectMath methods. These C++ functions might contain ObjectMath symbolic expressions which must be evaluated and expanded before compiling the C++ code, see [8] for an example. Translating

an ObjectMath model with C++ methods requires the following steps:

1. Translate the ObjectMath code into Mathematica code and load it into the Mathematica system.
2. Generate C++ class declarations and do some syntactic transformations on the user supplied C++ functions.
3. Generate C++ code for initialization of the external function interface.
4. Call Mathematica to evaluate ObjectMath expressions in the C++ code, once for each instance inheriting the C++ method. Expand the result from the symbolic evaluation into the C++ code.
5. Compile and link the C++ code.
6. Start the resulting program as a subprocess (computation server) of Mathematica.

The steps above are performed automatically. Any compilation errors in the C++ code are reported to the user.

4.7 Graphic Presentation

As mentioned earlier, some classes in the ObjectMath class library includes methods which generate three-dimensional pictures of bodies described with parametric surface techniques. Our parametric surface descriptions consist of a function of two arguments and intervals for the two parameters. A surface is obtained by varying the two parameters of the function. The generated picture can be combined with other graphical objects, for instance vectors representing forces and normals, axes in the coordinate systems, or textual labels. The user has control over several parameters concerned with the rendering of surfaces, such as lighting, shading and color. The view reference point may also be adjusted.

5 Related Work

There exist a number of systems and research areas which in some way are related to the ObjectMath programming environment. Some of these are:

- Computer algebra systems such as Maple [2] or Mathematica [11].
- Systems for matrix computations, e.g. MATLAB [7].
- Symbolic and numerical hybrid systems. An example is the FINGER package [9], a hybrid system supporting finite element analysis.

An exhaustive survey can be found in [3].

6 Conclusions

There is a strong need for efficient high-level programming support in scientific computing. The goal of our work has been to build an object-oriented programming environment that satisfies part of this need. A prototype environment supporting symbolic, numerical and graphic analysis has been implemented. The implemented programming environment has been successfully used in modeling and analyzing two different problems from the application domain (machine element analysis) in an industrial environment [4].

The successful use of the ObjectMath programming environment shows that a combination of programming in equations and object-orientation is suitable for modeling machine elements. Complex mathematical equations and functions can be expressed in a natural way instead of as low-level procedural code. The object-oriented features allow better structure of models and permit reuse of equations through inheritance.

References

[1] Joel F. Bartlett. Scheme→C, a portable Scheme-to-C compiler. Research Report 89-1, DEC Western Research Laboratory, Palo Alto, California, January 1989.

[2] Char, Geddes, Gonnet, Monagan, and Watt. *Maple Reference Manual.* WATCOM Publications, 5th edition, 1988.

[3] Peter Fritzson and Dag Fritzson. The need for high-level programming support in scientific computing applied to mechanical analysis. Technical Report LiTH-IDA-R-91-04, Department of Computer and Information Science, Linköping University, S-581 83, Linköping, Sweden, March 1991. Accepted for publication in *Computers and Structures – an International Journal.*

[4] Peter Fritzson, Lars Viklund, Johan Herber, and Dag Fritzson. Industrial application of object-oriented mathematical modeling and computer algebra in mechanical analysis. In Georg Heeg, Boris Magnusson, and Bertrand Meyer, editors, *Technology of Object-Oriented Languages and Systems – TOOLS 7.* Prentice Hall, 1992.

[5] Burton S. Garbow, Kenneth E. Hillstrom, and Jorge J. More. *Users Guide for MINPACK-1.* Argonne National Laboratory, Argonne, Illinois, USA, March 1980. Report ANL-80-74.

[6] Bil Lewis, Dan LaLiberte, and the GNU Manual Group. *The GNU Emacs Lisp Reference Manual.* Free Software Foundation, Inc., 675 Massachusetts Avenue Cambridge, MA 02139 USA, 1.02 edition, June 1990.

[7] Cleve Moler. MATLAB users' guide. Report CS81-1, University of New Mexico Computer Science Department, 1981.

[8] Lars Viklund and Peter Fritzson. An object-oriented language for symbolic computation – applied to machine element analysis. In Paul Wang, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, 1992.

[9] Paul S. Wang. FINGER: A symbolic system for automatic generation of numerical programs in finite element analysis. *Journal of Symbolic Computation*, 2:305–316, 1986.

[10] André Weinand, Erich Gamma, and Rudolf Marty. ET++ – an object-oriented application framework in C++. In *OOPSLA'88 Conference Proceedings*, 1988.

[11] Stephen Wolfram. *Mathematica – A System for Doing Mathematics by Computer.* Addison-Wesley Publishing Company, second edition, 1991.