International Journal of Soft Computing 2 (5): 595-598, 2007
ISSN: 1816-9503

Exloring Mathematical Software

Kostas Zotos
Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

Abstract: Today, it is not uncommon for software teachers to recommend that no function or method should be longer than a few lines. A few decades ago, the recommendation was the opposite: Don't put something in a separate subroutine if it is only called once. The reasons for this shift in software writing style are that mathematical projects have become bigger and more complex, that there is more focus on the costs of software development and that computers have become more powerful. The high priority of structured software development and the low priority of program efficiency are reflected, first and foremost, in the choice of programming language and interface frameworks. This study explores these considerations.

Key words: Mathematical software, programming in Mathematics, class design

INTRODUCTION

Making programs can be hard, but making code that is easy to maintain and extend is definitely hard, especially when the size of the program grows. Without careful planning and detailed specifications of the program it quickly becomes impossible to implement anything but the simplest program. As a consequence, different approaches to go from some problem through specification to an actual program exist and a plethora of programming languages have been created to aid the programmer to create correct programs faster.

It is widely admitted that traditional imperative programming languages such as FORTRAN or C are not the best suited for developing mathematical software. They present major lacks for extensibility, maintainability, or reusability, which are crucial objectives in designing libraries. Object-oriented design provides excellent opportunities to overcome limitations of traditional languages. Their major drawing power is to offer the opportunity to encapsulate complex coding and provide user-friendly interface (Pierre, 1994). My recommendation is to use the Java programming language. A strong driver for this decision is the widespread adoption of the Java language and its accompanying documentation and tools. Second, it supports safety, analyzability and flexibility.

STARTING NEW MATHEMATICAL SOFTWARE

Before starting new mathematical software, it is important to decide which programming language is best suited for the project at hand. Low-level languages are good for optimizing execution speed or program size, while high-level languages are good for making clear and well-structured code and for fast and easy development of user interfaces and interfaces to network resources, databases, etc.

It should be clear, that the choice of programming language is a compromise between efficiency, portability and development time. Interpreted languages are out of the question when efficiency is important. A language based on intermediate code and just-in-time compilation may be a viable compromise when portability and ease of development are more important than speed. This includes languages such as C#, Visual Basic .NET and the best Java implementations. However, these languages have the disadvantage of a very large runtime framework that must be loaded every time the program is run. The time it takes to load the framework and compile the program are often much more than the time it takes to execute the program and the runtime framework may use more resources than the program itself when running. Programs using such a framework sometimes have unacceptably long response times for simple tasks like pressing a button or moving the mouse. The NET framework should definitely be avoided when speed is critical.

The fastest execution is no doubt obtained with a fully compiled code. Compiled languages include C++, Pascal, Fortran and several other less well-known languages. C++ is supported by some very good compilers and optimized function libraries. C++ is an advanced high-level language with a wealth of advanced features rarely found in other languages. But the C++ language also includes the low-level C language as a

subset, giving access to low-level optimizations. Most C++ compilers are able to generate an assembly language output, which is useful for checking how well the compiler optimizes a piece of code. Furthermore, most C++ compilers allow assembly-like intrinsic functions, inline assembly or easy linking to assembly language modules when the highest level of optimization is needed. The C++ language is portable in the sense that C++ compilers exist for all major platforms. Pascal has many of the advantages of C++ but is not quite as versatile. Fortran is also quite efficient, but the syntax is very old-fashioned.

Development in C++ is quite efficient thanks to the availability of powerful development tools. One popular development tool is Microsoft Visual Studio. This tool can make two different implementations of C++, directly compiled code and intermediate code for the common language runtime of the NET framework. Obviously, the directly compiled version is preferred. An important disadvantage of C++ relates to security. There are no checks for array bounds violation, integer overflow and invalid pointers. The absence of such checks makes the code execute faster than other languages that do have such checks. But it is the responsibility of the programmer to make explicit checks for such errors in cases where they cannot be ruled out by the program logic. C++ is definitely the preferred programming language when the optimization of performance has high priority. The gain in performance over other programming languages can be quite substantial. This gain in performance can easily justify a possible minor increase in development time when performance is important to the end user.

There may be situations where a high level framework based on intermediate code is needed for other reasons, but part of the code still needs careful optimization. A mixed implementation can be a viable solution in such cases. The most critical part of the code can be implemented in compiled C++ or assembly language and the rest of the code, including user interface etc., can be implemented in the high level framework. The optimized part of the code can possibly be compiled as a Dynamic Link Library (DLL) which is called by the rest of the code. This is not an optimal solution because the high level framework still consumes a lot of resources and the transitions between the two kinds of code gives an extra overhead which consumes CPU time. But this solution can still give a considerable improvement in performance if the time-critical part of the code can be completely contained in a DLL.

PURE COMPUTER SCIENCE MEETS PURE MATHEMATICS

Computer science reuses techniques from engineering and Mathematics and adapts them to the new context. For example, Mathematics uses the concept of problem reduction that reduces one problem to an equivalent but a simpler problem that is more amenable to formal treatment. This technique is also very useful in computer science, for example, it is used to find computational thresholds or to convince someone that it is unlikely to find an efficient algorithm for certain problems. As an example, consider the set of equation systems where all equations are of the form x+y+z = 1. All variables are either one or zero and our goal is to find an efficient algorithm that satisfies the optimum fraction of the equations in a given equation system. The technique of problem reduction is used to show that the fraction 4/9 of the equations can be satisfied by an efficient algorithm. It is also used to show that it is unlikely that there is a better algorithm because the set of equation systems where the fraction 4/9 + e can be satisfied is NP-complete for arbitrary small e (Golden Ratio) (http://www.ccs.neu.edu/research/demeter/papers/mary-cs/mary4.PDF).

Students in computer science learn Object-oriented techniques as a method for designing software systems without explicitly knowing it they learn fundamental principles of abstract Mathematics. In Table 1, we are going to see that object-oriented principles are also very well suited for implementing "Mathematics" (Marc , 2003).

There are close conceptual links between Object-orientation and certain mathematical structures such as rings and groups (Marc Conrad, 2004). Figure 1 shows an UML diagram example. An abstract base class Ring requires from its child classes the implementation of the basic arithmetic (addition, multiplication, etc). A second class RingElt stores the information about the elements of a ring. Each RingElt instance a belongs to an instance of a child class R of the Ring. Examples for R are the ring of integers, rational numbers or a polynomial ring.

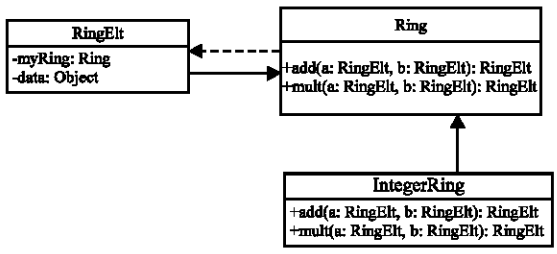
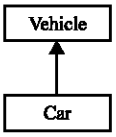
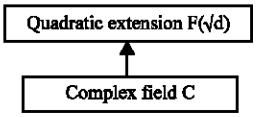
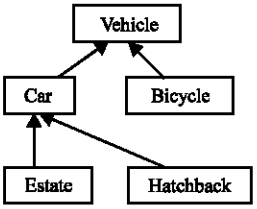
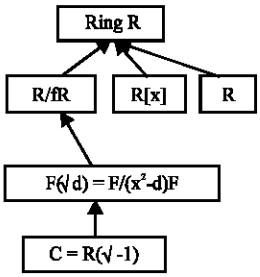


Fig. 1: Example of a ring

Table 1: Object-oriented principles are also very well suited for implementing "Mathematics"

	Object-oriented software	Math example
Inheritance: Give one class (the child class) all the methods and data of another class (the parent class): An "is a" relationship.	 The class car gets all methods of the vehicle + additional own methods.	 C = R(√-1) "is a" quadratic extension of R
Inheritance hierarchies usually have a tree or directed graph structure		
Overriding methods: Reimplement a method of the parent class in the child class.	Vehicle: Implement a method move(). Car (inherits Vehicle): Also implements a method move(). Car objects execute the move() method defined in the Car class.	F(√d): Implement multiplication C (inherits F(√d)): Re-implement multiplication, e.g. using polar coordinates in some cases.
An advantage of overriding methods: Generic algorithms	A class Driver can be associated with the Vehicle class. The Driver can move the Vehicle moving in fact a car or a bicycle.	Define a polynomial ring over an arbitrary ring of coefficients and obtain polynomials over C, Q, F(√d) etc.
Abstract methods abstract classes.	That is, the driver moves only Cars, Bicycles, but not a "Vehicle". In fact the Vehicle class does not need to define the move() method.	Similarly a Ring class does not need to define addition and multiplication. (But it is obvious, that a Ring has addition, multiplication, etc.)
Abstract methods: Declare a method in the parent class implement in the child class.	Vehicle: Declare a method move(). Car (inherits Vehicle): Also implements a method move(). Car objects execute the move() method defined in the Car class.	A Ring R: Declare multiplication. C (inherits R): Implement multiplication.

CONCLUSION

The very nature of mathematical software makes the designing of a system difficult. Unlike other areas of design, such as building construction and car manufacturing, the final product of software design is abstract and intangible. "It is not constrained by materials, governed by physical laws or by manufacturing processes" (Marc, 2004). This is both a positive and a negative characteristic of software. It is positive in the way that there are no physical limitations of what software can accomplish, yet n egative due to the fact that such systems can easily become largely complicated and difficult to comprehend. A commonality between designing software and designing other products is that there is no single correct solution. Given a particular item to develop, or task to complete, developers are faced with multiple design solutions where the one to implement is not always apparent. In addition, design involves many differing

dimensions, for example cost, reliability, maintainability and efficiency, all of which require optimisation. It is essential for the developer to find the right balance between these dimensions for their particular solution.

We are currently at a point in software development where structured, top-down design methods are no longer sufficient for handling the complexity of problems possible with today's hardware. We need to find a paradigm of software development that can handle the added complexity. University courses in programming nowadays stress the importance of structured and Object-oriented programming, modularity, reusability and systematization of the software development process. These requirements are often conflicting with the requirements of optimizing the software for speed or size. In mathematical academic environments software often seems to grow, without a clear plan or explicit intention of fulfilling some need or purpose, except perhaps as a vehicle for research.

REFERENCES

Marc Conrad, 2003. Abstract Classes-pure computer science meets pure mathematics.

Marc Conrad, 2004. Tim French. Exploring the synergies between the Object-oriented paradigm and mathematics: A Java led approach. Internatonal Journal of Mathematical Education Science Technology.

Pierre Manneback, 1994. Guibo Peng. Towards an Object Oriented Distributed Matrix Computation Library above C and PVM, Laboratoire P.I.P.

Sommerville, I., 2001. Software Engineering. Pearson Education Ltd.