CAS - New Strategies and Techniques

Kostas Zotos†

† *Department of Applied Informatics, University of Macedonia, Thessaloniki-Greece*

Summary
Computer algebra systems are software packages, usually Object-oriented, which are used in manipulation of mathematical formulas. The primary goal of a Computer Algebra System (CAS) is to automate tedious and sometimes difficult algebraic manipulation tasks. The specific uses and capabilities of these systems vary greatly from one system to another. Some of them include facilities for graphing equations and provide a programming language for the user to define their own procedures. In this paper we are going to examine new design strategies and techniques.

Key words:
Computer algebra systems; Mathematical software design; CAS

1. 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. Computer scientists, mathematicians and engineers often rely on Computer Algebra Systems (CASs) for computations requiring complicated calculations and their preference for one or another is due to the system's capability of solving the classes of problems of interest for such users. Yet, it is often desirable to be able to benefit from the functionality provided by external pieces of software, either of another Computer Algebra Systems or of the modules written in other programming languages. To achieve this desiderate the systems can be extended allowing the execution of external codes in a transparent way for the user.

Calculating technology in mathematics has evolved from four-function calculators to scientific calculators to graphing calculators and now to calculators (or computers) with Computer Algebra System (CAS) software. The advent of CAS software, which can do a great deal of the problems in a standard algebra or calculus text book at the push of a few buttons, truly represents a quantum leap in technology. The community of mathematics educators is in the throes of a great debate as to whether this is one of the most exciting or most frightening developments in the history of mathematics education as mathematics educators struggle with the implications of having software in the classroom which can, for example, expand and factorise algebraic expressions, solve equations, differentiate functions, and find anti-derivatives [1].

In the following sections, we are going to examine: Design principles of CAS (section 2), new strategies/techniques (section 3) and some results (section 4).

2. Design principles of CAS

There are major differences between mathematical libraries and CAS (Table 1). Both should consider four major goals (safety, analyzability, scalability and flexibility) in order to enable productivity and performance.

- **Safety.** Common errors, such as illegal pointer references, type errors, initialization errors, buffer overflows are to be ruled out by design.
- **Analyzability.** Both should be intended to be analyzable by programs (compilers, static analysis tools, program refactoring tools). Simplicity and generality of analyses requires that the programming constructs – particularly those concerned with concurrency and distribution – be conceptually simple, orthogonal in design, and combine well with as few limitations and corner cases as possible.
- **Scalability.** The scalability fundamentally depends on the properties of the underlying algorithms.
- **Flexibility.** It is necessary that the data-structuring, concurrency and distribution mechanisms be general and flexible.

It is widely admitted that traditional imperative programming languages such as FORTRAN or C are not the best suited for developing CAS. 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

Manuscript received July 5, 2007
Manuscript revised July 25, 2007

drawing power is to offer the opportunity to encapsulate complex coding and provide user-friendly interface [2]. Moreover, they support inheritance which allows one class (the parent class) to provide all its methods to a second class (the child class). This is also known as 'specialization' or an 'is a' relationship. Figure 1 shows how a simple inheritance relationship is defined between algebraic number fields. In this case the child classes comprise a cyclotomic number field and a quadratic number field [3].

Table 1. Libraries vs CAS

Libraries: • Extend mathematical capabilities of the language • Efficiency • Keep features of the language **CAS:** • Support special scientific computing features (symbolical computations, matrix computations etc.) • Convenience • Make use of graphics

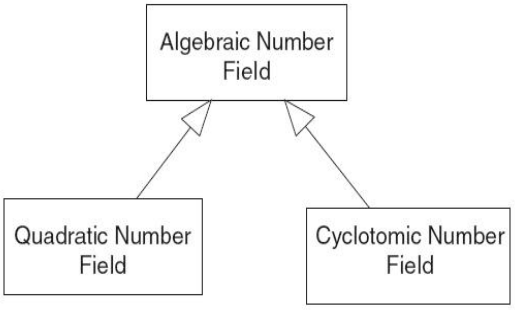

Fig.1 An inheritance hierarchy of algebraic fields.

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.

3. Experimental Consideration

The operations typically performed by traditional CAS are on data that has been abstracted from the original problem. For instance, one may solve a linear system in order to solve a system of Ordinary Differential Equations (ODEs). However, the fact that the linear system comes from ODEs is lost once the library receives the matrix and right hand side. By introducing metadata, we gain the facility of annotating problem data with information that is typically lost, but which may inform a decision making process. **Meta software** mediates between the application program and the computational platform so that application scientists (with disparate levels of knowledge of algorithmic and programmatic complexities of the underlying numerical software) can easily realize numerical solvers and efficiently solve their problem.

In finding the appropriate numerical algorithm for a problem we are faced with two issues:

1. There are often several algorithms that, potentially, solve the problem, and
2. Algorithms often have one or more parameters of some sort.

Thus, given user data, we have to choose an algorithm, and choose a proper parameter setting for it. **Self adapting strategy** in determining mathematical algorithms is the following:

• We solve a large collection of test problems by every available method, that is, every choice of algorithm, and a suitable 'binning' of algorithm parameters.
• Each problem is assigned to a class corresponding to the method that gave the fastest solution.
• We also draw up a list of characteristics of each problem.
• We then compute a probability density function for each class. As a result of this process we find a function $p_i(\bar{x})$ where i ranges over all classes, that is, all methods, and $\bar{x}$ is in the space of the vectors of features of the input problems. Given a new problem and its feature vector $\bar{x}$, we then decide to solve the problem with the method i for which $p_i(\bar{x})$ is maximised [4].

Figure 2 describes a meta-object procedure which focuses on three points: the easy reuse of cryptography-aware code, the composition of cryptographic services and the transparent addition of cryptography-based security to third-party code. The meta-object cryptography model for adding cryptography-based security has the following steps:

1. Load base-level classes.
2. Reflect about base-level classes. This means to create the meta configuration required by the base-level application.
3. Start up the meta objects from the secure initial state.
4. Load the classes of the base-level application.
5. Execute the base-level application from meta level.

Steps 1, 3 and 4, are the same for any application, having a few, parameterizable, differences. Steps 2 and 5 are what can vary among applications [5].

Mathematicians' and engineers' preference for one or another CAS encounters yet the drawback that even though CASs excel one another in solving selected classes of problems they do not offer a complete functionality. It would be therefore desirable to be able to augment them

with the capabilities of external systems. **Grid architecture** provides a solution to the problem. CAS can be integrated in Grid architecture and interact with external platforms (i.e. Java) [6]. This direction is followed in MathGridLink architecture [7]. The connection of MathGridLink with the Grid environment is bidirectional since either the deployment of Grid web services is possible from within Mathematica [8] or Grid. Two components, namely MathGrid Service Client (MGSC) and MathGrid Service Generator (MGSG), build up the system, as depicted in Figure 3. The first intermediates the access to any available Grid service, regardless the language this has been written in, allowing the invocation either in a blocking style in which Mathematica's kernel is entirely allocated for the service or in a no blocking manner such the a semaphore-based synchronization allows a multitasking in that the service invocation can be run in parallel mode. MGSG encapsulates a Java web service generator generating ready-to-deploy Grid archive, thus creating a web service interface. The resulted service deploys the Mathematica functions (specified by the developer) as Grid service methods in a transparent manner.

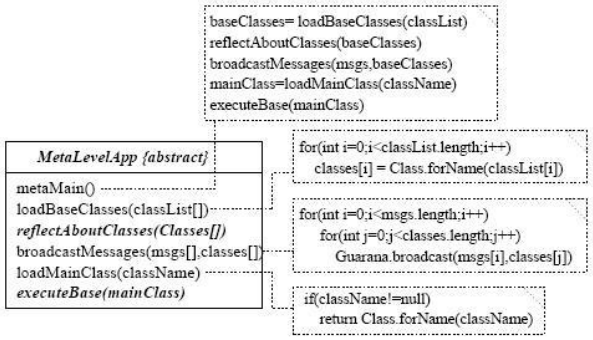

Fig. 2 A meta-object model

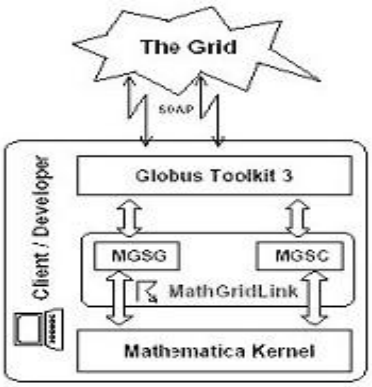

Fig. 3 MathGridLink middleware extending Mathematica to the Grid allowing the interaction with the Grid environment in two ways.

CAS programmers should try to detect and eliminate all fatal programming errors before the software is placed into service because, even if they can be detected efficiently, there probably isn't much that can be done about them after the application is placed into service except to log them on some sort of "black box" recorder and restart the application. Most of the math errors inherited from built-in types cannot be detected until run-time. Other programming errors cannot be detected until run-time because vector, matrix and tensor size information is not generally available at compile time and the one dimensional arrays which they reference must be allocated and deallocated dynamically. These errors include containment, range, conformance, reference and memory errors [9].

Classic algebraic libraries, based on imperative programming, contain sub algorithms for working with polynomials, matrices, vectors, etc. Their big inconvenience is the dependency on types. For example, a polynomial can be built over any kind of algebraic unitary commutative ring (R;+;*), and we have to define a different set of procedures that implement the common operations with polynomials, for every such ring. Object-oriented algebraic systems based on design patterns remove the inconvenience of type dependency [10]. Furthermore, allow us to build not only a flexible numerical algebraic system, but also a general abstract algebraic system. Finally, allow automatic conversions between compatible structures, and dynamic creation of new classes that correspond to different algebras specified by the user. For example, the Java Cryptography Architecture (JCA) specifies design patterns for designing cryptographic concepts and algorithms. The JCA architecture separates concepts from their implementations. These concepts are encapsulated by classes in the java.security and javax.crypto packages. These classes are called concept classes. JCA relies heavily on the factory method design pattern to supply instances of its concept classes. A factory method is basically a special kind of static method that returns an instance of a class [11]. Tile idea here is that a concept class is asked for an instance that implements a particular algorithm.

We propose to employ Aspect-oriented programming (AOP) at several points in the design of CAS. AOP can be used to decrease the amount of duplicated code, to check for special cases in a non-intrusive manner, and might even improve the performance of the system. A simple example would be the enforcement of domain-specific rules (e.g., for "add" methods that reside in disparate sub-classes). Using the additional "degree of freedom," more elaborate type systems (e.g., in the spirit of AXIOM) can be built. In particular, aspects could be used to define algebraic structures over data types that are defined by classes, while interfaces would provide for restriction and/or composition [12]. As applications are almost never

perfect in their first version, it is also possible for the developer to add some extra statements for debugging or tracing purposes. This introduces three natural aspects in this application: the multi-threading, the timing and the tracing. For example, the algorithm which multiplies the matrices A and B can be briefly described as follows:

1. *split the matrices A and B into slices,*
2. *multiply the slices together into sub-matrices of C,*
3. *store the sub-matrices into the final C matrix.*

This could easily fit into a single method. However, doing so would not make it easy to select the right join points at which to weave in the aspects. Even without envisaging an Aspect-oriented approach, writing everything in a single method does not favor the clarity of the code. Moreover, it is easier and cleaner to use method calls, as opposed to accesses to variables, as pointcuts. Indeed, if the code corresponding to the three steps of the multiplication algorithm described above is localized within a single method, defining pointcuts matching the join points between those steps cannot be done cleanly. Even if it is possible have pointcuts that select the accesses to some variable, it is not possible to clearly identify which part of the program flow is concerned when there is only a single method. Thus, it is sensible to write one method for each logical part of the above algorithm. Even though the component program should not be aware of the aspect program, it is possible to design the component program with some prior knowledge of the aspects.

4. 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” [13]. 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 negative 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.

The primary goal of my effort in CAS is to develop a new generation of algorithms and software packages. To succeed this, we should rethink the way that we build CAS. The issues to consider include software architecture, programming languages and environments, compile versus run-time functionality, data structures, and fundamental algorithm design.

References

[1] Meagher, M. “Learning in a Computer Algebra System (CAS) Environment”, Paper presented at the annual meeting of the North American Chapter of the International Group for the Psychology of Mathematics Education, 2004.

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

[3] Marc Conrad, Tim French. “Exploring the synergies between the Object-oriented paradigm and mathematics: a Java led approach”, International Journal of Mathematical Education in Science and Technology, 2004.

[4] “Self Adapting Numerical Software (SANS) Effort”, University of Tennessee and Oak Ridge National Laboratory, June 2005.

[5] Alexadre Braga, Richard Dahab. “A meta-object library for cryptography”, Relatorio Technico IC-1999-06.

[6] Diana Dubu. “Interconnecting Computer Algebra Systems within the Grid”, Master Thesis 2004.

[7] Tepeneu, D., Ida, T., “MathGridLink - A bridge between Mathematica and the Grid”, 2002.

[8] http:// www.wolfram.com/

[9] http://www.netwood.net/~edwin/svmtl/

[10] Virginia Niculescu, Grigoreta Sofia Moldovan. “Building an Object-oriented Computational Algebra System Based on Design Patterns”, INFORMATICA, Volume XLVIII, Number 1, 2003.

[11]Andrew Burnett, Keith Winters, and Tom Dowling, “Principles and Practice of Programming in Java”, pages 84, 85. National Umversity of Ireland 2002.

[12]Markus A. Hitz, “Aspect-Oriented Programming in the Design of Computer Algebra Libraries”, poster in Sigsam 2004.

[13]I. Sommerville. *Software Engineering.* Pearson Education Ltd., 2001.

Kostas Zotos is a research assistant in University of Macedonia (Department of Applied Informatics) under professor George Pekos and also supervised by assistant professor George Stephanides.