Reusable Reliable Software Components for Computer Algebra

Greg Butler

Centre Interuniversitaire en Calcul Mathématique Algébrique
Department of Computer Science
Concordia University
Montreal, Quebec, H3G 1M8 Canada
Tel: (514) 848-3031
Tel: (514) 848-3000
Fax: (514) 848-2830
Email: gregb@cs.concordia.ca

Abstract

Computer algebra is a good testbed for the study of reuse, and reusable libraries and frameworks should provide the tools to research software architectures for computer algebra systems. Computer algebraists demand correctness, reliability, robustness, and efficiency while insisting on ease of adaptability of the software. However, they do have a strong background in formalisms. The aim of our work is to construct reliable, reusable libraries and frameworks in C++ to support the development of experimental prototypes of next-generation computer algebra systems which will incorporate high-level object-oriented languages, databases, deduction, and planning. There is a particular emphasis on the use of formal methods for specification and verification.

Keywords: Reuse, formal methods, library, framework, computer algebra

Workshop Goals: Learning; networking;

Working Groups: reuse and formal methods, reuse and OO methods, reusable component certification, design guidelines for reuse — C++

1 Background

The position outlined here derives from a very real need for reliable, reusable and efficient software components in order to perform research into software architectures for the next-generation of computer algebra systems. Such systems will integrate features from database and knowledge base systems, theorem provers, planners, and scientific visualisation with features from existing computer algebra systems, such as extremely complex algebraic algorithms. Most likely the user language of the next-generation systems will combine the paradigms of functional, logic, object-oriented, and set-based languages. Such a system would provide effective reuse of domain knowledge to novices and experts alike, where the domain is some subset of mathematics such as algebra, combinatorics, or group theory.

Cayley/Magma is one computer algebra system developing in that direction. It is being developed at the University of Sydney under the leadership of John Cannon. The Cayley project [1] began in 1975. It was implemented in Fortran, and concentrated on computational group theory. In 1987 it was converted to C almost fully automatically. Development has continued in C. The Cayley team has — and has had from the very beginning — a strong emphasis on design and implementation *with* reuse and *for* reuse, because of the very limited manpower. Since 1974, as part of this team, I have worked on all aspects of software for algebraic algorithms: conceptualisation, proof of correctness, complexity analysis, data structures, design and implementation, performance measurement and tuning. My software is the most heavily used part of the Cayley system. One suite of routines [2], for backtrack searches, written in 1978, would now be called a framework. In 1986 John Cannon and I started work on Cayley, version 4 [3], an ambitious step towards a fully integrated next-generation system for discrete algebraic and combinatorial computation. I focussed on the design of the user language, and on the knowledge base facilities. My student, Sridhar Iyer, and I built the first deductive database of groups [4, 5], and a prototype knowledge base of groups [6, 7], both using NU-Prolog. NU-Prolog provides a foreign-function interface to our C routines which perform the required algebraic computations for the knowledge base. The impedance mismatch between mathematical objects, external databases, and Prolog is a major issue.

These experiences convinced me that a more flexible environment, using a single language, C++, is needed to research the issues of software architectures and their integration for computer algebra systems. Also, there is a great need from researchers in algebraic algorithms for a software library. Since moving to Concordia University at the end of 1991, my research has focussed on C++ libraries and frameworks, issues of reuse, and issues of reliability (which includes correctness).

2 Position

My position can be summarised by the following three points.

(1) Software Engineering researchers need very close links with real-world software practitioners and applications; in our case with builders of computer algebra systems.

(2) Evaluation of reusable libraries and frameworks needs attempts at actual reuse of them; in our case by builders of computer algebra systems, and developers of algebraic algorithms.

(3) Our principal obstacles to reuse are understanding, trust, and efficiency.

We address understanding through a combination of documentation, formal specification, and proofs of correctness (at various levels of formality).

We address trust through a combination of the above, together with suites of test cases, and a history of effective reuse.

We address efficiency through the use of specialised algorithms, hybrid algorithms, and layered multi-lingual implementations (which allows the use of C and machine code).

While there is a consensus amongst the reuse community over understanding and trust being obstacles to reuse, efficiency is rarely stressed. Our reusers insist on it, as their scientific applications may be at the frontier of what can be achieved with available computer resources.

The research plan we follow is to:

- Identify existing algorithms, data structures, components, and tools used in language translators, databases, and computer algebra systems (e.g. NU-Prolog, Cayley, ISOM, and Algeb). In particular, review existing libraries such as Leda, COOL, Interviews, NIHCL, mat++, newmat.
- Select those to reverse engineer (into C++) based on the interests of current students, related projects, and prototypes under development.
- Insist upon high quality documentation, formal specification, extensive reasoning about correctness, and extensive test suites for the components developed for the libraries.
- Identify efficiency bottlenecks using performance benchmarks and seamlessly eliminate them through use of hybrid algorithms, algorithms for special cases, and the use of C and machine language routines for critical operations (as done in the bignum library [8]).
- Assimilate libraries written in very high level languages (VHLL) (such as Algeb, Cayley) by constructing translators into C++ and provide the necessary run-time support. The translation must allow the reuse of existing library components for operations or procedures written in the VHLL.
- Preferably have the components in use as part of a related project being carried out concurrently with the development of the library components.

Reusability is validated and evaluated by developing small *demonstration prototypes* of applications. For example, current work is concentrating on

Framework for deductive databases A framework for Datalog is being implemented in C++. It incorporates several mechanisms for multi-dimensional database retrieval. Furthermore, a formal specification in Z of a relational database and a deductive database is underway. The demonstration prototype will be to implement the *TwoGroups* database in C++ using the framework.

Framework for Backtrack Searches It is planned to reverse engineer the existing framework(s) into C++. The demonstration prototype will be to implement the *ISOM* package for combinatorial enumeration in C++ using the framework.

Translator for Algeb The translator from Algeb to C++ is being implemented in C++. Library components for arithmetic with arbitrary precision integers, reals, p-adic numbers, vectors, matrices, and polynomials are being implemented in C++. We plan to use Larch/C++

to specify the library components. The demonstration prototype will be to (automatically) translate the Algeb implementation of the LLL algorithm into a highly efficient version coded in C++ (and possibly C and machine code).

3 Comparison

The only comparable computer algebra system to Cayley/Magma is Axiom. Axiom has a sophisticated type system, and good visualisation tools. However, being more concerned with continuous mathematics, it does not include database and knowledge base facilities.

There has been several frameworks developed [9], the best known being Smalltalk's MVC framework and the Choices framework for operating systems. Don Batory at University of Texas at Austin has developed a framework for traditional database systems [10], and is in the process of extending it to object-oriented database systems. His approach views a database as a composition of functional layers (or realms), and the framework as the realms, their type constraints as functions, and the alternative implementations. This is different from the usual view of a framework as a collection of cooperating abstract classes (and concrete subclasses). There has been no work on object-oriented frameworks for deductive databases, for Prolog, for knowledge bases, or for computer algebra systems.

There are several well documented libraries. Some, such as BigNum [8] and Leda [11], go as far as providing pre- and post conditions as part of the interface documentation. To our knowledge, only the work of Musser and Stepanov [12, 13] reasons carefully about the correctness of the library. Work on local certifiability [14] and language constructs which support formal reasoning about reusable components [15] by Weide and his colleagues also treat correctness issues, but more generally.

Much more work is required on documenting, specifying, and reasoning about frameworks, thus extending [16, 17, 18].

References

[1] J. Cannon, "An Introduction to the Group Theory Language, Cayley," in *Computational Group Theory* (M. Atkinson, ed.), pp. 145–183, London: Academic Press, 1984.

[2] G. Butler, "Computing in Permutation and Matrix Groups II: Backtrack Algorithm," *Math. Comp.*, vol. 39, pp. 671–680, 1982.

[3] G. Butler and J. Cannon, "Cayley, Version 4 : The User Language," in *Symbolic and Algebraic Computation. Proceedings of 1988 International Symposium on Symbolic and Algebraic Computation, Rome* (P. Gianni, ed.), (Berlin), pp. 456–466, Springer-Verlag, July 4–8 1989. Lecture Notes in Computer Science, 358.

[4] G. Butler, S. Iyer, and S. Ley, "A Deductive Database for the Groups of Order Dividing 128," in *ISSAC 91* (S. Watt, ed.), (New York), pp. 210–218, ACM Press, 1991.

[5] G. Butler, S. Iyer, and E. O'Brien, "A Database of Groups of Prime-Power Order." submitted to ACM TODS.

[6] G. Butler and S. Iyer, "Deductive Mathematical Databases — A Case Study," in *Statistical and Scientific Database Management. Proceedings of 5th International Conference on Statistical and Scientific Databases, Charlotte, North Carolina* (Z. Michalewicz, ed.), (Berlin), pp. 50–64, Springer-Verlag, April 3–5 1990. Lecture Notes in Computer Science, 420.

[7] G. Butler and S. Iyer, "An Experimental Knowledge Base of Simple Groups." in preparation.

[8] B. Serpette, J. Vuillemin, and J. Hervé, "BigNum: A Portable and Efficient Package for Arbitrary-Precision Arithmetic," Tech. Rep. Paris Research Laboratory Report 2, Digital Equipment Corporation, 1989.

[9] R. Johnson and B. Foote, "Designing Reusable Classes," *Journal of Object-Oriented Programming*, vol. 1, pp. 22–35, 1988.

[10] D. Batory and S. O'Malley, "The Design and Implementation of Hierarchical Software Systems with Reusable Components," *ACM Trans. on Software Engineering and methodology*, vol. 1, no. 4, pp. 355–398, 1992.

[11] K. Mehlhorn and S. Näher, "LEDA, a Library of Efficient Data Types and Algorithms," Tech. Rep. TR A 04/89, FB10, Universität des Saarlandes, Saarbrücken, 1989.

[12] D. Musser and A. Stepanov, "Generic programming," *LNCS*, vol. 358, pp. 13–25, 1989.

[13] D. Musser and A. Stepanov, *The Ada Generic Library: Linear Data Structure Packages.* Berlin: Springer-Verlag, 1989.

[14] B. Weide and J. Hollingsworth, "Scalability of Reuse Technology to Large Systems Requires Local Certifiability," in *Proceedings of the Fifth Annual Workshop on Software Reuse* (L. Latour, S. L. Philbrick, and M. Stevens, eds.), October 1992.

[15] D. Harms and B. Weide, "Swapping vs Copying: Their Influence on the Design of Reusable Software Components," *IEEE Transactions on Software Engineering*, vol. 17, pp. 424–435, 1991.

[16] R. Helm, I. Holland, and D. Gangopadhyay, "Contracts: Specifying Behavioral Compositions in Object-Oriented Systems," in *ECOOP/OOPSLA'90*, pp. 169–180, 1990.

[17] R. Johnson, "Documenting Frameworks Using Patterns," in *OOPSLA'92*, pp. 63–76, 1992.

[18] W. Cook, "Interfaces and Specifications for the Smalltalk-80 Collection Classes," in *OOPSLA'92*, pp. 1–15, 1992.

4 Biography

Gregory Butler is Associate Professor in Computer Science and a member of Centre Interuniversitaire en Calcul Mathématique Algébrique (CICMA) at Concordia University, Montreal, Canada. He obtained his PhD from the University of Sydney in 1980 for work on computational group theory. He spent 1976/77 at ETH, Zürich as part of his doctoral studies, and for two years, 1979–1981, was a postdoctoral fellow at McGill and Concordia Universities in Montreal. He was on the faculty of the Department of Computer Science at the University of Sydney from 1981 to 1990. He has held visiting positions at the University of Delaware and Universität Bayreuth.