Parallel Computer Algebra Software as a Web Component

Andreas Weber* Wolfgang Küchlin
Bernhard Eggers
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen, 72076 Tübingen, Germany
E-mail: {weber,kuechlin}@informatik.uni-tuebingen.de
WWW: http://www-sr.informatik.uni-tuebingen.de

Abstract

In the field of computer algebra, joining separate computer algebra systems with individual strengths into problem solving environments is an important recent research direction. In this paper we describe how we wrapped a specialized computer algebra system, which uses shared-memory parallelism to speed up Gröbner basis computations, into a Web component that can be used by other systems. By this component also a new algorithmic development in Gröbner basis computations—the use of the *Fractal Gröbner Walk*—was made available to various general purpose computer algebra systems in the form of a single optimized implementation. Previously, all of the important algorithmic improvements to an algorithm had to be reimplemented in the general purpose systems. Our implementation allows access to the Gröbner basis computations via "intelligent" graphical user interfaces and via scripting from existing general purpose algebra systems.

1 Introduction

Joining separate numeric and algebraic systems and algorithmic software components with individual strengths into problem solving environments is an important recent research direction. Currently, the existing systems can be subdivided roughly into two categories.

General purpose systems — algebra systems such as Axiom, Macsyma, Magma, Maple, Mathematica, Mupad, Reduce or numeric systems such as Matlab, Octave, Scilab — come with nice user interfaces and sophisticated graphical capabilities and contain a broad range of computer algebra or numeric algorithms. However, because of the size of the systems and the broad range they cover, they cannot be always up to date algorithmically. Also the software architecture of such a general purpose system—consisting of a kernel and an interpreted language in which many algorithms are written—yields computational overheads that can be avoided by systems optimized for a certain class of algebra problems. Also these systems are in general not suited to make use of special hardware, such as shared-memory multiprocessors.

Specialized systems are usually developed by research groups that concentrate on algorithm research in one special area of computer algebra or scientific computing. These systems usually contain the latest algorithms of these areas and their implementations are very often tuned up considerably. Several of these systems are developed for high-performance computing architectures such as parallel machines. However, these systems usually have a limited scope and crude user interfaces.

Connections between different algebra systems—or algebra systems and other scientific code—are of major interest for different reasons.

First, instead of reimplementing a new algorithm in all of the general purpose systems, a single optimized implementation could be used, if the specialized system in which the algorithm is implemented, and which runs e. g. on a parallel machine, could be reused as a *software component*.

Second, the different areas of computer algebra overlap and many of the fundamental algorithms developed in one area can be used in other areas, too. As a result, in many specialized systems many algorithms have to be reimplemented because they are needed for further extensions. This reimplementation is then often done by a group specialized in another area, who might not implement the best available algorithm but only a well known one.

Third, solving problems that arise in "real world situations" usually requires a combination of methods, not only from computer algebra but also from other areas.

Below we will describe how we achieved such a combination of systems in the case of a high performance Gröbner basis solver which can use the power of shared-memory multiprocessors. Thus this software will perform best on special hardware and should not be ported to all platforms on which a potential user works.

The encapsulation of the parallel system and access to it is done with Java using various techniques that are well supported by Java. We use the MathBus [18] exchange format for the transmission of mathematical data that has been developed over the last years. We can easily adapt our software to another exchange format such as OpenMath [11] or can use several formats simultaneously. Having such an exchange format is one important prerequisite for the combination of various systems. Exchange formats address the data aspect of the communication between systems. The orthogonal issue of calling the functionality of a remote system has to be treated by different means. For this purpose we found the various techniques offered by Java very useful—provided the remote functionality is coarse grained enough so that the performance penalties given by Java for the communication are negligible.[1] So we assume that our solution is prototypical for many other systems, too.

The organization of the paper will be as follows: In Sec. 2 we will give some background information about Gröbner basis computations in general and the high-performance Gröbner solver which we made accessible in particular. In Sec. 3 we describe the

*Supported by *Deutsche Forschungsgemeinschaft* under grants We 1945/1-1 and Ku 966/4-1.

[1] For low or medium grained applications the use of a CORBA object broker is advantageous. We refer to [17] for a case study which shows that CORBA object brokers can be used to efficiently call remote functionality of a surprisingly low granularity.

various access methods we provided and their technical realization. In the concluding Section 4 we discuss various aspects of the use of Java for our project that will apply to similar projects.

2 Background

2.1 Gröbner Bases

The computation of Gröbner bases [6, 7, 5] is of central importance in computer algebra. They have many applications ranging from pure mathematics to the solution of systems of non-linear polynomial equations arising, for example, in the inverse kinematic problem in robotics. Full sections of conferences like ISSAC [14, 15, 16], or even entire conferences [9] are devoted to the topic of Gröbner basis computations. Thus we can only give a rough sketch of the topic and refer to the literature for further details.

Given a set $\mathcal{P}$ of (in general non-linear) polynomials, a Gröbner basis is a unique normal form $\mathrm{GB}(\mathcal{P})$ for $\mathcal{P}$ with the same roots. The shape of $\mathrm{GB}(\mathcal{P})$ depends on an ordering on the monomials, the so-called term ordering. Different orderings are useful for different purposes; e. g. total degree orderings usually lead to fast Gröbner basis computations, lexicographic orderings lead to systems from which the actual solutions can be extracted relatively easily.

Given any $\mathcal{P}$ and term order $>$, Buchberger's famous algorithm will compute $\mathrm{GB}_>(\mathcal{P})$ in finite (but possibly astronomic) time. Given any $\mathrm{GB}_>(\mathcal{P})$ and another term order $\succ$, the *Gröbner walk algorithm* [10, 4] will compute $\mathrm{GB}_\succ(\mathcal{P}) = \mathrm{Walk}_\succ(\mathrm{GB}_>(\mathcal{P}))$ in finite time. Practical experience shows that it is often much more efficient to compute $\mathrm{Walk}_{\mathrm{lex}}(\mathrm{GB}_{\mathtt{tot}}(\mathcal{P}))$ than $\mathrm{GB}_{\mathrm{lex}}(\mathcal{P})$. Experience has also shown that Buchberger's algorithm is extremely difficult to parallelize.

The speed of a Gröbner basis computation heavily depends on several other parameters, whose choice has been the topic of intense research. The result of the computation will be independent of the choice of these internal parameters. However, they usually have a dramatic influence on the necessary computation time. There is no best choice for any of these internal parameters, but there are examples for which each of the possible settings performs better than the others. However, some heuristics have been developed that will yield good results for many examples. We included one of these heuristics into our system, cf. Sec. 3.1.

Each of the general purpose computer algebra systems has an implementation of Gröbner basis algorithms, but because of the algorithmic developments these implementations perform worse in general than the recent high-performance implementations. To our knowledge the so called *Fractal Gröbner Walk* [1]—an improved Gröbner walk procedure—has not been integrated into one of the general purpose systems up to now. On the one hand, since the computation time for Gröbner bases given by application problems very often ranges from several minutes or hours to intractably large, fast Gröbner basis algorithms are important. On the other hand—since the computation of a Gröbner basis is very often only one step in a series of computations—the integration of a fast Gröbner basis engine into a larger environment is also important.

Thus the general situation sketched in the introduction applies particularly well to Gröbner basis computations.

2.2 High Performance Gröbner Basis Computation

The high-performance Gröbner solver that we made available via the WWW is an improvement of the system described in [2]. It is based on the parallel computer algebra nucleus PARCAN [12]. This nucleus gives a very efficient implementation of arbitrary precision integers on the basis of GNU/MP and of lists, including an automatic garbage collection. Using the parallel features of PARCAN the Gröbner solver shows remarkable speed-ups on shared-memory multiprocessors. Even super-linear speed-ups due to strategy effects have been observed [2]. It currently runs on multiprocessor SUN SPARC stations.

On the algorithmic side, improvements of the Gröbner walk technique have been developed [3, 4] and implemented. So far the most efficient of these is the *Fractal Walk* [1], which is the version we use in the system described in this paper.

3 Accessing the High-Performance Gröbner Basis Software via the WWW

We provide three different access methods to our high-performance symbolic software.

Methods that allow scripting access to the high-performance solvers out of a general purpose algebra system are one of them. This method of access will be described in Sec. 3.2.

A graphical user interface in the form of a Java applet will be more convenient for casual users. Using copy and paste operations data can be transferred from an application into the data input Window of the applet. The format of various algebra systems can be used for input and output to the applet. Support for other formats can be added dynamically by a user, if converters to the new target system and our internally used format are provided. The graphical user interface in this sophisticated version uses several features of JDK 1.1. We will describe it in more detail in Sec. 3.1.

To allow access to the system from older browsers we still export a simpler graphical user interface that uses JDK 1.0.2.

The graphical user interfaces can be reached from the following URL:
`http://www-sr.informatik.uni-tuebingen.de/projects/webcomponents/`
If you are interested in the software for scripting access, write an e-mail to the first author.

3.1 A Sophisticated Graphical Java Interface

For a description of the functionality of this user interface, we refer to the URL given above. From this Web page, the Java applet building the graphical user interface itself and some commenting Web pages can be reached. Below we will focus on some important points in its internal organization.

Dealing with Different Representations of Mathematical Data. Different computer algebra systems often use different syntactic representations of the mathematical objects they are dealing with. Since input of data into our graphical user interface will be typically achieved by copy and paste operations of textual representations of mathematical expressions out of a general purpose computer algebra system, we support several formats for input and output of mathematical data.

Internally we use a single exchange format and provide parsers and converters from and to other formats. The exchange format we use is the MathBus [18], which was especially developed as a bridge between different systems, especially computer algebra systems. However, we can easily adapt our software to another exchange format such as OpenMath [11] or can use several formats simultaneously.

Up to now we have implemented converters from and to the computer algebra systems Maple and Mathematica and the format used by the parallel Gröbner solver. Converting data from the MathBus into another format is quite easy, since the MathBus format is a quite general expression tree format. Converting data from Mathematica into the MathBus format could be done with a parser generator.[2] We used the Java Bison system [13] for this purpose. Incorporating converters for other algebra systems—such as Mupad, Macsyma, or Reduce—can be done by simply adapting the grammar of the parser for the algebra system and writing the straight-forward conversion from the MathBus format into the format of the new system. So an extension to many other formats is straight forward.

The converter classes all inherit from a common class, and the choice of the actually used converter class is done via the dynamic class selection and load mechanism of Java. The user can choose a converter via selection from a list or by providing its name. Thus an unlimited number of new converters can be included into the graphical user interface dynamically, if the appropriate naming conventions are obeyed.

Choice of a Good Computation Strategy. The result of a Gröbner basis computation is independent of the value of several *internal parameters* but their appropriate choice usually has a dramatic influence on the computation time. Many users who need the result of a Gröbner basis computation but are not researchers in the field of Gröbner bases will find it valuable if the system provides a good heuristic choice for these internal parameters. Thus we have included the computation of such a heuristic into our applet. Having the full power of an object-oriented language such as Java available at the WWW-interface, this is a relatively easy task. However, using a simpler interface such as an HTML-form, this useful functionality would be hard to provide.

Internal Organization of the Different Computations. A schematic view of the internal organization is given in Fig. 1. The graphical user interface, the functions that select a computation strategy and the converters from the different data formats are all written in Java and will be performed on the client computer.

The following programs will be executed on our parallel server computer. The parallel Gröbner solver; a high-performance implementation of the Fractal Gröbner Walk; a small C program based on the SACLIB computer algebra library [8] which expands polynomials. This function serves as an improvement to the user friendliness of the interface, because the user might give polynomials in non-expanded form, but the Gröbner solver expects its input polynomials in expanded form. These programs are wrapped into Java classes. The connection between the applet residing on the client and the classes on the server side is achieved via RMI. For performance reasons the first two of these programs should always run on a high-performance computer. The last of these functions could be included into the applet. However, using an existing C-library the required functionality could be implemented with little effort in C, whereas porting the needed functionality from the SACLIB to Java would have required a much greater effort. Thus this functionality also is assigned to the server machine. Wrapping it as a Java class and making it available to the applet via RMI gave almost the same functionality as a direct Java implementation and allowed the reuse of proven C-code.

RMI allowed a very nice integration of this wrapped high-performance code into the Java code written for the client. Besides the interaction of the user with the GUI, the general data flow is that the input data are parsed into MathBus format according to the user specified parser. The input polynomials are normalized by the component that expands polynomials, and the result is used as input to the Gröbner solver, together with the user input of the term ordering induced by the ordering on variables specified by the user. Moreover, this function also gets the "internal parameters" as input, which were either chosen by our heuristics or specified by the user. The return value of this function is the desired result unless a Gröbner Basis with respect to the lexicographic ordering has to be computed using the Fractal Gröbner Walk. In this case, the return value is the input to the Gröbner walk component, together with the parameters for start ordering and target ordering, which are known to the program.

This data flow requires several calls to the remote objects together with some minor computations in the client. The implementation of the code for these computations and the required calls to the remote objects could be written almost as if only local calls to Java objects had to be performed. Especially the treatment of exceptional states—parse errors in the converters written in Java, out of memory exceptions for the remote components or other special states such as a failure in a Gröbner basis computation—could be done uniformly by the Java exception mechanism, independent of their status as local or remote calls. For this purpose the Java wrapper for the C-programs examine and transform diagnostic output of the C-programs into Java exceptions, which are then transported transparently to the client by RMI. The output of the C-programs which constitute the result of the computation is wrapped into the return value of the Java wrapper function.

Thus this program infrastructure is well suited to be enhanced by additional software components—e. g. ones that solve a system of non-linear equations via a Gröbner basis computation. The program complexity for connecting such remote components is not higher than using local Java method calls in the implementation of a new Java method.

3.2 Scripting Access to the High-Performance Software

Although access via one of the graphical user interfaces described above is much more convenient for a casual use of the parallel software, frequent users might want to have scripting access to our parallel system out of their own algebra systems.

We thus provide Java classes that allow the various general purpose systems to encapsulate the remote access to the parallel Gröbner solver in a function of the particular system. To overcome the security restrictions on applets these classes have to be installed on the client computer. Most of the client side classes of the sophisticated graphical user interface (cf. Sec. 3.1) are reused for this purpose, and via RMI they access the *identical server side software* as the sophisticated graphical user interface does. A schematic view of these software components is given in Fig. 2.

If a host algebra system allows direct access to Java, these classes can be incorporated directly. However, the current general purpose algebra systems do not have this feature. Usually they allow system calls and communication via pipes. Thus for the client side proxies we have implemented wrapper Java programs that read their input from standard input, write the result to standard output, and write the values of any exceptions to standard error. In addition to the parameters of the algebraic functions these programs also take the name of an algebra system as an additional parameter and perform the appropriate conversions for the input and output data.

Using these client side Java programs we will provide functions for the computer algebra systems Maple and Mathematica, which behave to the user in the same way as the built-in Maple resp. Mathematica functions for Gröbner basis computations do, but which use the scripting access to our parallel Gröbner solver and the Fractal Gröbner Walk component to perform the computations instead of using a local sequential computation. Since Maple and Mathematica allow system calls and communication via pipes,

[2] The same holds for the format used by Maple or the parallel Gröbner solver.

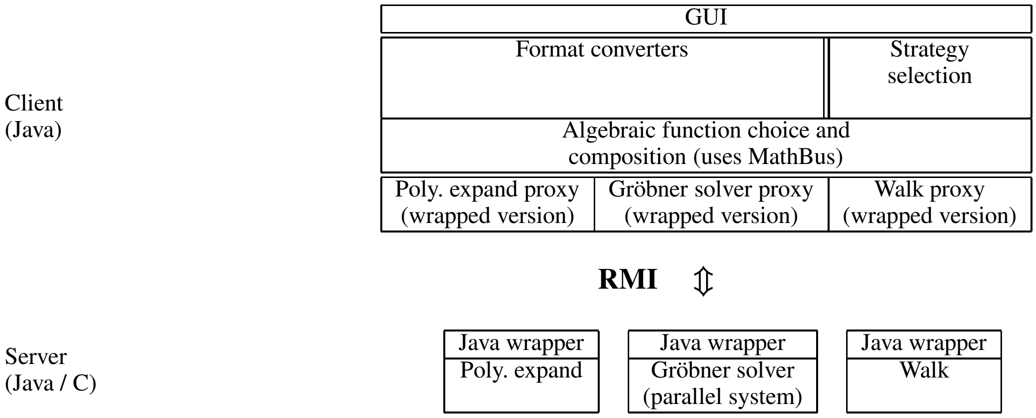

Figure 1: Schematic overview of the software components: graphical user interface

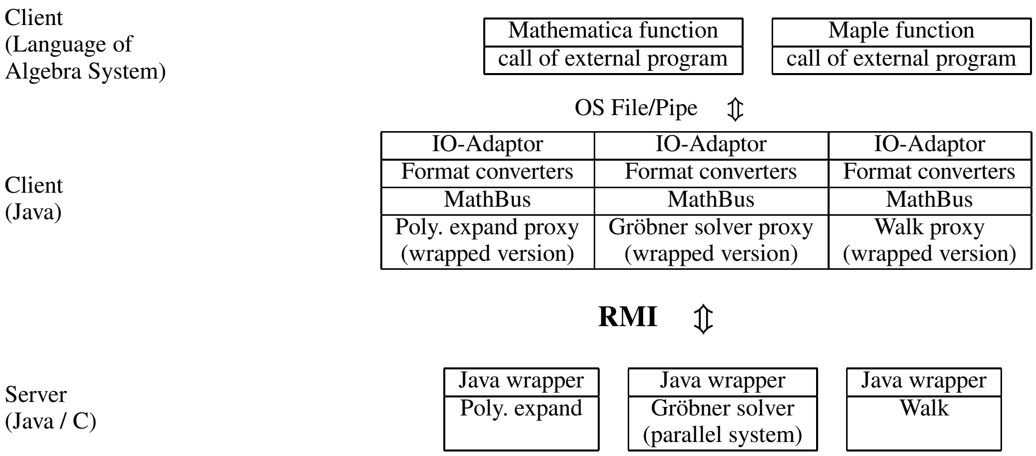

Figure 2: Schematic overview of the software components: scripting access to general purpose system

the client side Java programs can be accessed from the Maple resp. Mathematica functions in this way. The client side Java programs access the server functionality via RMI in the same way as it is accessed from the graphical user interface. The heuristics to choose the internal parameters will be computed in the Maple resp. Mathematica functions we provide.

3.3 Use of Java Beans

So far we have not provided our Java components as Java Beans but as simple applets and stand alone Java programs and Java class libraries. Using Java Beans for a graphical user interface would have the advantage that frequent users could store their own default settings instead of using the ones provided by us. However, there are few default settings independent of the computation of a heuristics for the choice of the internal parameters. Changes in these heuristics cannot be easily accomplished by the user by the currently available Java Beans mechanisms. Thus up to now the use of Java Beans seems to have little advantages over a simpler Java applet for our particular application.

Moreover, we think that for frequent users the access method of choice is to use the scripting interface out of their general purpose algebra systems. Up to now the use of Java Beans gives little advantages for the scripting interface: some of our functions need an additional parameter which could be avoided using Java Beans, but this is a minor issue in the few lines of code that have to be written once for any target algebra system.

As soon as the general purpose computer algebra systems provide direct access to Java and Java Beans, the use of Java Beans will be reconsidered. Nevertheless, it takes only a relatively minor effort to transform our components to Java Beans.

4 Conclusion

The idea of software components is certainly not a new one. E. g., the success of UNIX can be attributed in part to the concept of many small software tools that can be combined to more powerful ones by pipes or shell scripts.

However, Java based software components have all the advantages of UNIX components but also give several important enhancements over the UNIX concept. The arguments to a UNIX software component are usually given in the string of command line arguments. Thus no static argument type checking can be done, but all typing errors in the arguments of a UNIX components can only be detected at run time. Wrapping software components—such as ours—in Java programs gives the full type checking support of a Java compiler to detect type errors at compile time.

Another very important advantage of Java as a wrapper language is the possibility to use the Java exception mechanism to reliably communicate exceptional states of our software components—such as parse errors, out of memory exceptions, or resource limitations that we have imposed on remote users.

Moreover, Java as a scripting language is much more powerful than the scripting languages for UNIX shells or similar ones that had to be used otherwise. The support of the object-oriented paradigm, which has proven to be so fruitful for many larger applications, is missing in most of the traditional scripting languages.

The use of a single programming language for implementing or wrapping additional software components such as the parser classes and as a scripting language to connect different components has several additional advantages.

- It is not necessary that the implementer is proficient in several languages in order to produce high-quality code.
- A much better interaction between the different components is possible, if they are written or included within the same programming language.
- Since the user interface is also written in Java, a much better integration between the software and the user interface is possible. For instance the intelligent selection of a good computation strategy out of user requirements would have been much harder to realize if the user interface and the scripting language that connects the software components had been different.

The most important aspect for our purpose was the availability of RMI. Although we also implemented access to the Gröbner system and to the Gröbner walk in our JDK 1.0.2 interface in a more traditional client server system—using socket communication and cgi-scripts—the integration of several remote components into one new system was greatly facilitated by RMI. Connecting several remote components—the Gröbner solver, the program computing a Fractal Gröbner Walk, or the small component that expands polynomials—together with some additional functionality written in Java—such as the converter classes—can be done in a clear software structure using RMI.

With RMI the wrapped high-performance C-code could be used within Java as if it had been written in Java. Thus there is no need to port the high-performance code to other architectures. Since high-performance code is often optimized towards a specialized architecture, porting this code to other architectures is often not only difficult but would often yield a suboptimal solution. In this respect we suppose that our solution is not only prototypical for high-performance computer algebra code, but also for any high-performance code.

Moreover, this tight integration of code into Java gave the flexibility not only to access the high-performance code on a remote shared-memory multiprocessor, but also to reuse some legacy code as a component, which is not bound to a specialized hardware.

Since the parallel Gröbner code should run with the highest possible performance but any invocation of it is of quite coarse granularity, wrapping the existing code and accessing it via RMI currently has a much better performance characteristics than translating all of the Gröbner code into Java.

Acknowledgments. We are grateful to B. Amrhein and O. Gloor for many helpful discussions about Gröbner basis computations. The software kernels that we wrapped to components, connected via Java, and made available on the WWW were built within their research on high-performance Gröbner basis computations, supported by *Deutsche Forschungsgemeinschaft* under grant Ku 966/2-2. We are grateful to R. Zippel for many helpful discussions about the MathBus.

References

[1] AMRHEIN, B., AND GLOOR, O. The Fractal Walk. In Buchberger and Winkler [9], pp. 305–322.

[2] AMRHEIN, B., GLOOR, O., AND KÜCHLIN, W. A case study of multi-threaded Gröbner basis completion. In Lakshman [15].

[3] AMRHEIN, B., GLOOR, O., AND KÜCHLIN, W. Walking faster. In *Design and Implementation of Symbolic Computation Systems — International Symposium DISCO '96* (Karlsruhe, Germany, Sept. 1996), J. Calmet and C. Limongelli, Eds., vol. 1128 of *Lecture Notes in Computer Science*, Springer-Verlag.

[4] AMRHEIN, B., GLOOR, O., AND KÜCHLIN, W. On the Walk. *Theoretical Comput. Sci. 187* (1997), 179–202.

[5] BECKER, T., AND WEISPFENNING, V. *Gröbner Bases: A Computational Approach to Commutative Algebra*, vol. 141 of *Graduate Texts in Mathematics*. Springer-Verlag, 1993.

[6] BUCHBERGER, B. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, Universität Innsbruck, 1965.

[7] BUCHBERGER, B. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems. *Aequationes Mathematicae 4*, 3 (1970), 374–383.

[8] BUCHBERGER, B., COLLINS, G. E., ENCARNACIÓN, M. J., HONG, H., JOHNSON, J. R., KRANDICK, W., LOOS, R., MANDACHE, A., NEUBACHER, A., AND VIELHABER, H. *SACLIB User's Guide*. Johannes Kepler Universität, 4020 Linz, Austria, Mar. 1993. Available via anonymous ftp at `melmac.risc.uni-linz.ac.at` in `pub/saclib`.

[9] BUCHBERGER, B., AND WINKLER, F., Eds. *Gröbner Bases and Applications—Int. Conf. 33 Years of Gröbner Bases* (Feb. 1998), vol. 251 of *London Mathematical Society Lecture Notes Series*, Cambridge University Press.

[10] COLLART, S., KALKBRENER, M., AND MALL, D. Converting bases with the Gröbner walk. *Journal of Symbolic Computation 24* (1997), 465–469.

[11] DALMAS, S., GAËTANO, M., AND WATT, S. An OpenMath 1.0 implementation. In Küchlin [14], pp. 241–248.

[12] GLOOR, O., AND MÜLLER, S. PARCAN—a parallel computer algebra nucleus. In preparation, 1998.

[13] HEIMBIGNER, D. *JAVA Bison—Parser and Lexer Generation for Java*. University of Colorado at Boulder, 1997. Available at `ftp://ftp.cs.colorado.edu/pub/cs/distribs/arcadia/jb.tar`.

[14] KÜCHLIN, W., Ed. *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (ISSAC '97)* (Aston Wailea Resort, Maui, Hawaii, 1997), The Association for Computing Machinery, ACM.

[15] LAKSHMAN, Y. N., Ed. *Proc. Intl. Symposium on Symbolic and Algebraic Computation (ISSAC '96)* (Zürich, July 1996), Association for Computing Machinery.

[16] LEVELT, A. H. M., Ed. *Proc. Intl. Symposium on Symbolic and Algebraic Computation (ISSAC '95)* (Montreal, July 1995), Association for Computing Machinery.

[17] WEBER, A., KÜCHLIN, W., AND HOSS, J. Connecting computer algebra systems via CORBA—a case study. Submitted to ISSAC '98.

[18] ZIPPEL, R. *The MathBus Term Structure*. Cornell University, Ithaca, NY 14853, U. S. A., 1997. `http://www2.cs.cornell.edu/Simlab/papers/mathbus/mathTerm.htm`.