A Parallel Gröbner Solver as a Web Component

Andreas Weber* Wolfgang Küchlin Bernhard Eggers
Volker Simonis

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

Joining separate computer algebra systems and numeric 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. The architecture is prototypical for many other specialized systems to be joined into problem solving environments.

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.

*Supported by *Deutsche Forschungsgemeinschaft* under grant Ku 966/4-1.

General purpose systems such as Axiom [21], Magma [10], Maple [11, 12], Matlab [24] Mathematica [30], Reduce [19], or Scilab [26] 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 by them, e. g. on a parallel machine, if the specialized system in which the algorithm is implemented could be reused as a *software component* by a general purpose system.

Second, the different areas of computer algebra overlap strongly 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, which might not implement the best available algorithm but only a well known one.

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

Fourth, the user can use a component that runs on a high-performance computer and use the power of such a machine for a possibly application critical subtask. Since such subtasks are very often coarse grained, remote execution of them is often possible and advantageous, when it is possible to use the power of high-performance hardware on a remote execution site.

Below we will describe how we achieved such a combination of systems in the case of a high performance Gröbner bases 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 one of the exchange formats for the transmission of mathematical data that have been developed in the last years.[1] 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

[1]Currently, we are using the MathBus [31]. However, we can easily adapt our software to another format such as OpenMath [14] or can use several formats simultaneously.

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 such that the performance penalties given by Java for the communication are negligible.[2] 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 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

We will recall some basic facts about Gröbner bases [7, 8, 6] which will be needed in our system description below.

A Gröbner basis is computed for a set of polynomials with respect to a *term ordering* induced by an ordering on the indeterminates of the polynomials.

For many applications a Gröbner basis with respect to the so called lexicographic term ordering has to be computed. However, in many cases the computation of such a Gröbner basis is much more cost intensive than the computation of a Gröbner basis with respect to another ordering such as the total degree ordering or grevlex ordering, followed by a conversion of the basis with respect to this ordering to a basis with respect to the lexicographic ordering. The existence of good conversion algorithms has been shown only quite recently; one of these methods is the *Gröbner walk* [13, 4].

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 *Gröbner walk* has not been integrated into one of general purpose systems up to now. On the one hand—since the computation time for Gröbner basis given by application problems is very often ranging from several minutes or hours to intractably large—fast Gröbner basis algorithms

[2]For low or medium grained applications the use of a CORBA object broker is advantageous. We refer to [28] for a case study which shows that CORBA object brokers can be used to efficiently call remote functionality of a surprisingly low granularity.

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 Bases Computation

The high-performance Gröbner solver [2] that we made available via the WWW is based on the computer algebra nucleus CANNES [18, 15]. CANNES gives a very efficient implementation of arbitrary precision integers on the basis of GNU/MP and of lists, including an automatic garbage collection. The Gröbner solver uses the parallel version PARCAN [15] of CANNES and shows good speed-ups on shared-memory multiprocessors. Even super-linear speed-ups due to strategy effects have been observed [2]. It currently runs on multi-processor 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 Bases Software via the WWW

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

- A graphical user interface in the form of a Java applet. 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 he provides converters to the new target system and our internally used format. 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. This interface will be described in Sec. 3.2.

- Methods that allow scripting access to the high-performance solvers out of a general purpose algebra system. This method of access will be described in Sec. 3.3.

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.

3.1.1 Dealing with Different Representations of Mathematical Data

Different computer algebra systems often use different syntactic representations of the mathematical objects they are dealing with. Several suggestions have been made for a common exchange format between different systems. These include the *multi-protocol* (MP) and its extensions [16, 5], OpenMath [14], or the MathBus [31].

Ideally, all systems should support one of these formats. Then the communication between different systems would be greatly facilitated. Several systems have started to support OpenMath, which might be the common standard in the coming years. However, only a part of the systems we are interested in support this exchange protocol up to now.

Our solution to this problem has been as follows. Instead of being fixed on one exchange format, we just take one of the common exchange formats for our internal representation and provide parsers and converters from and to other formats. These other formats might be native formats of an algebra system or one of the other exchange formats the particular algebra system supports. Since for many purposes—including ours—providing conversions between the different exchange formats is a more or less straight-forward task, the choice of the exchange format is not crucial for us. Having one internal exchange format has the advantage that the problem of connecting n different systems is reduced from a quadratic problem to a linear one.

The main exchange format we currently use is the one of the MathBus. Up to now we have implemented a converter from and to the computer algebra system Mathematica and the format used by the parallel Gröbner solver. A converter from and to Maple will follow shortly. Converting data form 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.[3] We used the Java Bison system [20] for this purpose. Incorporating converters for other algebra systems 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 quite easy.[4]

The converter classes all inherit from a common class, and the choice of the actually used

[3] The same holds for the format used by the parallel Gröbner solver.

[4] The subset of Maple expressions that have to be handled for our application can be easily converted into MathBus format by the Java Bison system. Thus using this system for Maple in the same way we did for Mathematica is the direction we will take for the near future. Since OpenMath is supported by Maple and there is an OpenMath Java library [25] a detour via an OpenMath / MathBus conversion might be advantageous for a longer perspective. Since by our object-oriented software architecture the conversion from native Maple format into MathBus format is encapsulated, a change of the implementation from one method to the other does not affect other parts of the system.

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.

3.1.2 Choice of a Good Computation Strategy

Although the result of a Gröbner basis computation is independent of the internal parameters mentioned in Sec. 2.1 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 an heuristics 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.

The choice given by our heuristics will give the fastest computation in many cases. However, the computation of Gröbner bases is a difficult topic. So for any setting there are examples for which another choice would yield a faster computation. So each value can be overwritten by an expert user. Moreover, if a Gröbner basis with respect to the lexicographic term ordering has to be computed, the default setting will be to compute the one with respect to the grevlex ordering and to use the fractal Gröbner walk to compute the Gröbner basis for the lexicographic ordering.

Currently, the following rules are implemented in our applet. When the target ordering is as follows, the applet will suggest the following setting.

lexicographic: Compute a Gröbner basis with respect to the grevlex-order and use the fractal Gröbner walk to compute the Gröbner bases for the lexicographic ordering.

If the user insists on using a direct computation then use the *sugar strategy* for pair selection. The polynomial selection will be by weight, and the basis ordering by age.

total degree: Directly compute the Gröbner basis. Use the *least common multiple strategy* for pair selection. The polynomial selection will be by weight, and the basis ordering by age.

grevlex: The selection of the internal parameters is the same as for total degree.

In all cases the parallel implementation will be chosen, and the number of reduction workers will be set to four, because currently the computation will be performed on a 4-processor shared-memory SUN workstation.

3.1.3 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.

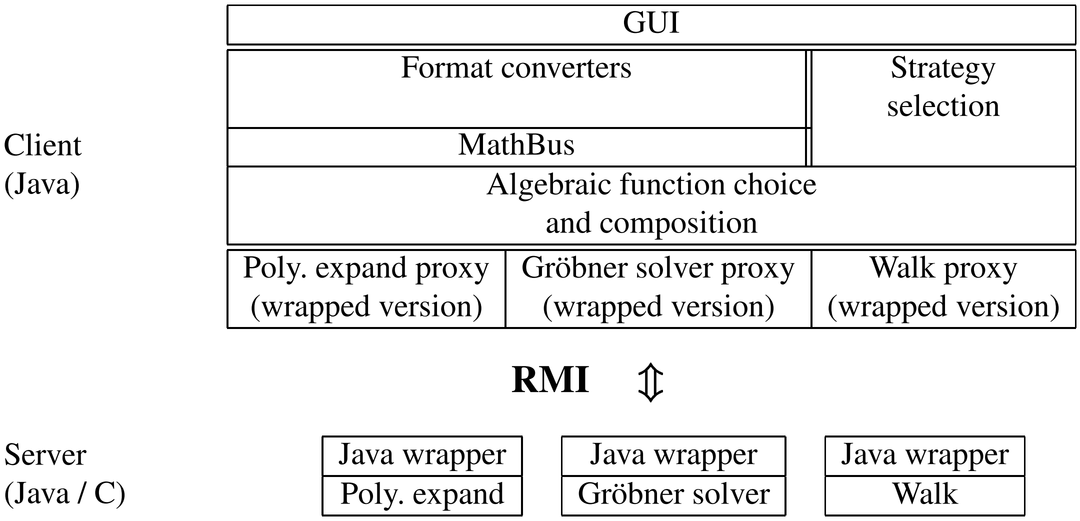

Figure 1: Schematic overview of the software components

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 [9] 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 [27].

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 high-quality 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, an 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 specialized by the user. The return value of this function is the desired result unless a Gröbner Basis with respect to the

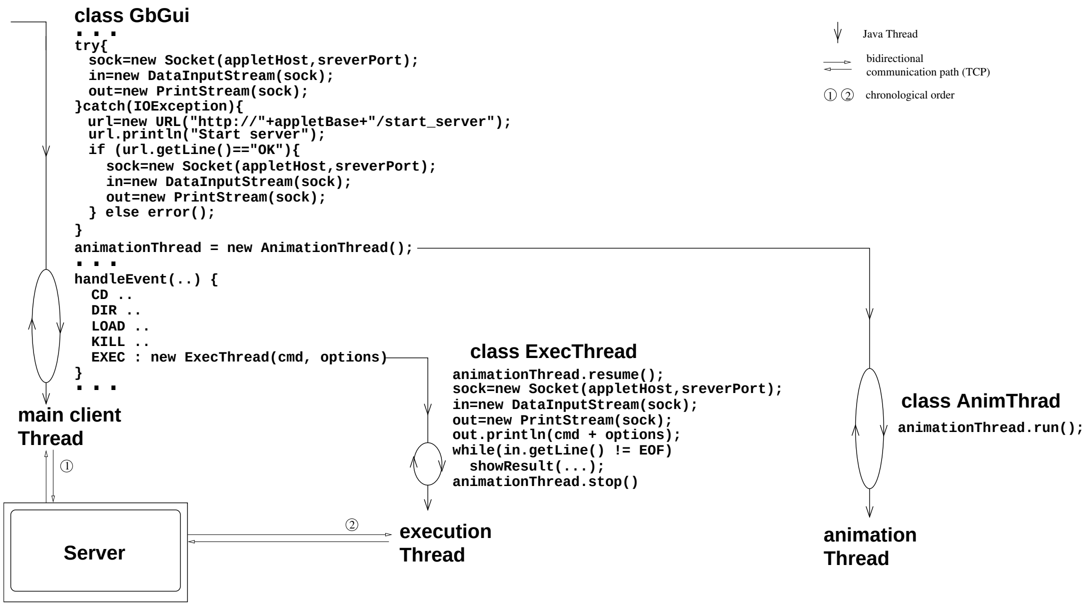

Figure 2: Client side program flow for simple GUI

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 then are 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 A Java Front-End for Client-Server Access

This front end is superseded by the sophisticated one described in Sec. 3.1. However, we will provide it as long as a significant number of browsers does not support JDK 1.1 functionality.

This front-end is based on a more traditional client-server concept, which is also well sup-

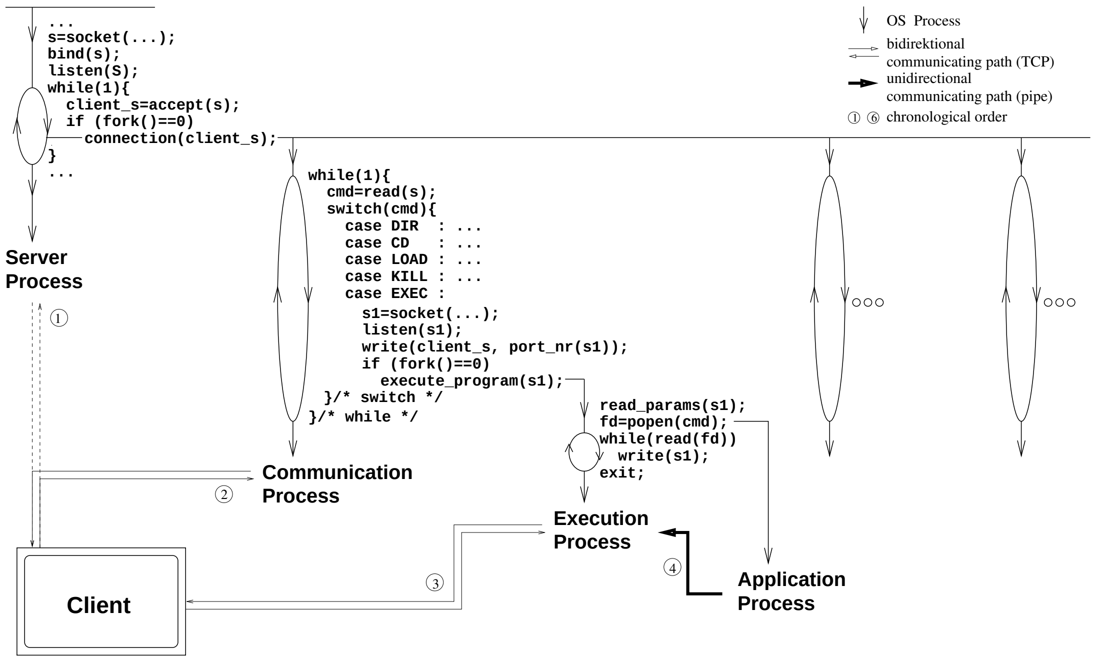

Figure 3: Server side program flow for simple GUI

ported by the Java class library.

The GUI for this front-end is a Java applet. A schematic view of the client side of the communication is given in Fig. 2.

On the server side, a program written in C is started as a cgi-script, which starts a socket communication with the Java applet. Via this communication commands and data are exchanged between the client and the server. For the exchange of commands we have implemented a small protocol of our own.

This server program also starts the Gröbner basis program on a potentially remote host in our server domain—since our WWW server is not necessarily a suitable high-performance machine. It will communicate and moderate the data and commands between the GUI and the computing engine. The program flow on the server side is schematically given in Fig. 3.

This interface is mainly intended for an expert user. It gives full access to all internal parameters. It also gives the possibility to load several examples described in the literature as input data, which might be used to compare the effects of different settings of the internal parameters.

We have provided two separate interfaces for the Gröbner basis computation engine and the Gröbner walk engine; the latter gives also full access to all settings of the Gröbner walk program and allows also experiments with conversions that are of interest to the expert user only.

3.3 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.

If a host algebra system allows direct access to Java, these classes can be incorporated directly. However, most of the current systems will not have this feature. Thus the common data exchange will be over IO-streams. We have implemented a sample Java program that reads its input from standard input, writes the result to standard output, and the values of any exceptions to standard error. If the host algebra system allows socket communication, this program is easily adapted to it.

We are currently working on the implementation of 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 function for Gröbner basis computation do, but which will 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.

3.4 Use of Java Beans

Currently, we have not provided our Java components as Java Beans [17] but as simple applets and stand alone Java programs and Java class libraries. Using Java Beans for graphical user interface had 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 the computation of a heuristics for the choice of the internal parameters. Changes in this heuristics cannot be easily accomplished by the user by the currently available Java Beans mechanisms. Thus up to now the use of Java Beans seem to have little advantages over a simpler Java applet for our particular application.

Moreover, we feel 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 has 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, then the use of Java Beans is worth being 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 the Gröbner solver, the program that computes a fractal Gröbner walk, or also a small component such as the one that normalizes the input polynomials—in Java programs gives the full type checking support of a Java compiler to detect type error at compile time.

Another very important advantage of Java as a wrapper language that we found 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 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 high-performance numeric 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.

Taking all these points together we found that Java is a very good choice to connect high-performance scientific code into problem solving environments.

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 and connected via Java, and made available on the WWW were to a great extend built within their research on high-performance Gröbner basis computations. This research is supported by the *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 Winkler [29].

[2] AMRHEIN, B., GLOOR, O., AND KÜCHLIN, W. A case study of multi-threaded Gröbner basis completion. In Lakshman [23].

[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] BACHMANN, O., SCHÖNEMANN, H., AND GRAY, S. MPP: A framework for distributed polynomial computations. In Lakshman [23], pp. 103–111.

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

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

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

[9] 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`.

[10] CANNON, J., AND PLAYOUST, C. *Algebraic Programming with Magma*. Springer-Verlag, 1997. See also `http://www.maths.usyd.edu.au:8000/comp/magma/Overview.html`.

[11] CHAR, B. W., GEDDES, K. O., GONNET, G. H., BENTON, L. L., MONAGAN, M. B., AND WATT, S. M. *Maple V Language Reference Manual*. Springer-Verlag, New York, 1991.

[12] CHAR, B. W., GEDDES, K. O., GONNET, G. H., BENTON, L. L., MONAGAN, M. B., AND WATT, S. M. *Maple V Library Reference Manual*. Springer-Verlag, New York, 1991.

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

[14] DALMAS, S., GAËTANO, M., AND WATT, S. An OpenMath 1.0 implementation. In Küchlin [22], pp. 241–248.

[15] GLOOR, O., AND MÜLLER, S. PARCAN—a parallel computer algebra nucleus. In preparation, 1997.

[16] GRAY, S., KAJLER, N., AND WANG, P. MP: A protocol for efficient exchange of mathematical expressions. In *Proc. Intl. Symposium on Symbolic and Algebraic Computation (ISSAC '94)* (Oxford, July 1994), Association for Computing Machinery, pp. 330–335.

[17] HAMILTON, G. *JavaBeans*. Sun Microsystems, July 1997. For Version 1.01. `http://java.sun.com/beans/`.

[18] HAUG, H. A guide to CANNES. In preparation, 1997.

[19] HEARN, A. C. REDUCE user's manual, Version 3.3. Report CP 78, The RAND Corporation, Santa Monica, July 1987.

[20] 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`.

[21] JENKS, R. D., AND SUTOR, R. S. *AXIOM: The Scientific Computation System*. Springer-Verlag, New York, 1992.

[22] 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.

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

[24] MATHWORKS INC. Matlab 5, 1997. http://www.mathworks.com/products/matlab/.

[25] POLYMATH DEVELOPMENT GROUP. *Java OpenMath Library*. Centre for Experimental and Constructive Mathematics, Simon Fraser University, Burnaby, BC, Canada, 1997. Version 0.3. http://pdg.cecm.sfu.ca/openmath/lib/.

[26] SCILAB GROUP. Scilab home page, 1997. http://www-rocq.inria.fr/scilab/.

[27] SUN MICROSYSTEMS. *Java Remote Method Invocation — Distributed Computing for Java*, 1997. http://java.sun.com:80/marketing/collateral/javarmi.html.

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

[29] WINKLER, F., Ed. *33 Years of Gröbner Bases* (Feb. 1998), Cambridge University Press.

[30] WOLFRAM, S. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, Redwood City, CA, 1991.

[31] 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.