

An Implementation of Distributed Mathematical Services

Stephan M. Hess, Christoph G. Jung, Michael Kohlhase, Volker Sorge
FB Informatik, Universität des Saarlandes, Germany
<http://www.ags.uni-sb.de>

Abstract

Real-world applications of theorem proving require open and modern software environments that enable modularization, distribution, inter-operability, networking, and coordination. This paper describes the *DMS* architecture for automated theorem proving that connects a wide-range of *mathematical services* by a common, *mathematical software bus*. It also presents an implementation, *Oz-DMS*, of the architecture in the Oz programming language. *Oz-DMS* provides the functionality to turn existing theorem proving systems and tools into mathematical services that are homogeneously integrated into a networked proof development environment. The environment thus gains the services from these particular modules, but each module in turn gains from using the features of other, plugged-in components.

1 Introduction

The work reported in this paper originates in the effort to develop a practical mathematical assistant system that integrates external deductive components. The Ω MEGA-system [BCF⁺97] is an interactive, plan-based deduction system with the ultimate goal of supporting theorem proving in main-stream mathematics and mathematics education. To provide the necessary reasoning and symbolic computation facilities it incorporates the first-order theorem provers OTTER [MW97], SPASS [Wei97], PROTEIN [BF94], the higher-order theorem provers TPS [ABI⁺96] and LEO [BK98], and finally the experimental Computer Algebra system μ CAS [KKS98].

Traditional deduction systems, such as the ones integrated into Ω MEGA, as well as today's tactical theorem provers, such as ISABELLE or NQTHM, are monolithical systems. They either work like compilers – reading a problem file and writing proof and log files after successful computation – or like programming environments featuring their own command interpreter or graphical user interface. Driven by the complexity of real-world reasoning problems and practical considerations in designing and interacting with the system, we have seen a rapid move towards integrative frameworks combining various external reasoners [Den93, HKK⁺94, Dah97] and computation systems [HT93b, BHC95, HT93a, KKS98].

Ideally, the reasoning modules in the Ω MEGA system interact with each other to complete open subgoals during the development of a proof. This can be initiated and supervised on-line by the user. This can be also guided by the Ω MEGA system itself, for instance during proof planning in order to expand a given proof plan to a full proof. Unfortunately, it is not always clear in advance, which prover is best suited for the problem at hand. Furthermore, the user could be asked to support the system with additional knowledge. Thus, Ω MEGA will call several ‘services’ in parallel in order to maximize the likelihood of success and minimize the time the user has to spend waiting for the system. In any case, the proofs found by these systems are transformed into the internal proof format of the Ω MEGA system; again this proof transformation process should run in parallel to the ongoing user interaction.

The role of the mathematical assistant in particular, but also of general applications of theorem proving in the large, for instance in program verification applications such as the VSE system [HLS⁺96] also developed in our group, call for an open and distributed system architecture. In such an architecture, the developer of a deduction system or a respective tool upgrades it

to a so-called *mathematical service* (see section 3) by providing it with an interface to a common *mathematical software bus* (see [Hom96, HC96b] for details) and therefore rather provides the mathematical service instead of the software itself.

In the context of the Ω MEGA system, we have implemented and experimented with such a network architecture, where the integrated theorem proving systems and tools interact distributed over the Internet and can be dynamically added to and subtracted from the coordinated reasoning repertoire of the complete system. Using these experiences as a running example, the possible benefits of a distributed approach to semi-automated proof development are fourfold:

Modularization The more external reasoners a system like the Ω MEGA system integrates the heavier the burden of installing and maintaining them gets. For instance, the kernel of Ω MEGA alone is a rather large system (roughly 17 MB of COMMON LISP (CLOS) code for the main body in the current version), its successful installation depends on the presence of (proprietary) LISP compilers or interpreters. This situation is similar for the other reasoning systems integrated into the Ω MEGA system, which come from numerous different original sources. For the user it is a difficult task to install and understand the complete system, for the developers it is a tedious task to port the system to more commonly used (free) compilers. Thus providing a mathematical service instead of software will free the developer from this task at the cost of providing the hardware resources. It will ease the maintenance of the particular modules and environments build upon them and it encapsulates related functionality into re-usable components.

Independence Deduction systems are among the most complex existing AI programs, they are typically developed by more than one individual and the respective components require vastly differing specialized know-how that is nowadays impossible to acquire for a single individual. The equivalent is true for Computer Algebra systems that exist in a vast variety from multipurpose to very specialized ones. Both user and developer can hardly distinguish which system is best suited for a particular task, let alone be able to use all different systems. Thus a distributed architecture of mathematical services with well-specified interfaces allows the focussed and independent development in specialized research groups, specialized application areas, and in specialized techniques.

Interoperability Along with independent development, the requirement appears of being able to easily put together a complete, and working system out of those components. Having a common platform of exchanging services across the network makes these components interoperable. Thus, components provide additional functionality into the system as a whole and, in turn, are provided with additional services in order to perform their service far more efficient.

Performance Finally, the performance aspect of theorem proving in the large is addressed by a distributed architecture. In local computer networks the situation is quite common that users have relatively low-speed machines on their desktop, whereas some high-speed servers that are accessible for everyone operate in the background. Running, e.g., the user interface on the local machine uses the local resources that are 'close to the relevant data' and sufficient for this task while the more powerful servers can be fully exploited for the really complex task of actually proving theorems.

Indeed, the vision in which we would like to extend the existing architecture and implementation goes further. Such mathematical services as we have proposed could be as well modeled as self-interested, autonomous software *agents* interacting in a common, also human-inhabited environment. Following the definition of [WJ95], they are *reactive* in the sense that they are steadily interacting with users or other software agents working on the same proof. They are *pro-active* in that they adopt and autonomously work on particular goals in performing their service. And they are *social* in the sense that they request other agents or even the human user to support the successful execution of their service. Thus theorem proving appears to be a joint effort of a society of communicating, independent agents.

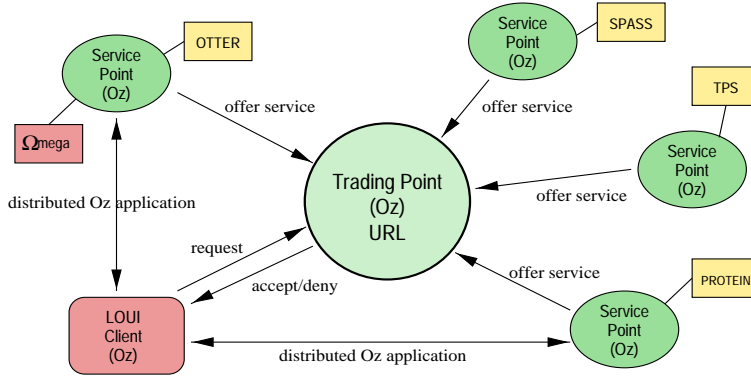


Figure 1: The architecture for distributed mathematical services.

The purpose of the present paper is to motivate and document a first step towards the practical realization of such a perspective. It is not to give a formal semantics for the integration of mathematical services via a mathematical software bus (see [Hom96, HC96b] for a discussion).

In the remainder of the paper, we will therefore give an overview of the network architecture of the Ω MEGA system developed so far (section 2) and discuss the technical realization of the prototype implementation *OZ-DMS* (section 2.2). Then we will categorize existing (and planned) mathematical services by their behavior and communication needs (section 3). Finally in section 4, we will develop a proposal for a deduction meta-language as a network communication layer that will support an agent-based extension of *DMS*.

2 Ω MEGA as a Distributed Proof Development System

In this section we will describe the network architecture currently realized for the Ω MEGA system (see figure 1). The mathematical services consist of various automated theorem provers and the *LOUI* user interface. The computer algebra system μ CAS is not realized as a mathematical service, since it is so small, that the administration overhead outweighs the possible benefits. The move to a generic network architecture for Ω MEGA was motivated as a preparation for the integration of the computer algebra system MAGMA [CP98].

The software bus functionality in the implementation is realized by the trader model, which limits the central administration of the bus to a so-called *trading point* that provides routing and authentication information to the mathematical services. we will describe this model in more detail in the next section.

2.1 The Trader Model for a Mathematical Software bus

The central application in the trader architecture is the so-called **tradingpoint**, an administrative service that is reachable by other applications via an URL. The tradingpoint's main task is to administer the different kinds of mathematical services. If the same service is available on different hosts, then it also has to decide which host is used to serve the client's request by monitoring the work load on the different machines.

The different mathematical services on different hosts are managed via **servicepoints**. This is an application, which manages all available services available on the host. To keep administration overhead low, there is only one **servicepoint** per host. To make the different services available to the world, the servicepoint offers them to the tradingpoint. The servicepoints have also the task to supervise the clients and restrict their accesses to the concrete services.

A client that wants to use a certain service with a special functionality requests for that service at the **tradingpoint**, this accepts or denies the request depending on whether the service

is available or not. If it is, the `tradingpoint` establishes a private communication channel between the client and the service, which these can use without further interaction with the `tradingpoint`. In the trader model, the client does not need to know where the service is running nor how it is implemented, but only the trader's URL and the expected functionality of the service. Figure 1 gives an overview.

2.2 Oz-DMS, an implementation of the Trader Model

Oz-DMS is realized via a distributed programming system, called MOZART. MOZART is an interactive and distributed implementation of the concurrent, logic-based programming language Oz [Smo95], and it will be released to the general public mid-1998. Besides state-of-the-art object-oriented programming, its powerful inference engines, and a neat support for building reactive user interfaces, it provides the full infrastructure to write distributed applications. Its main strength comes from its network transparency and network awareness which make it easy to 'agentify' arbitrary applications.

Network transparency means that the semantics of Oz programs does not change if you distribute computations among different sites. For example, the programmer can use lexical scoping, logical variables, objects, etc. in distributed applications.

Network awareness means that the programmer has full control over the network operations. The language provides mobile and stationary objects, i.e. methods are executed locally (the object moves) or remotely (the message moves). The programmer has control over structure copying among sites. Structures may be copied eagerly or lazily.

The choice of this programming system has been central to the realization, since it has provided all the necessary programming features. The fact that MOZART also provides high-level inference primitives like constraint propagation and search makes it a good implementation choice for the mathematical services proper. An example of a mathematical service that is fully implemented in MOZART is the generic proof visualization tool *LQUT*.

Distributing Ω MEGA Up to this point, we have considered a client-server network with one server that is dedicated to Ω MEGA itself and several clients that use this server. In reality, a Ω MEGA network may consist of several servers that can be accessed via a gateway service. The gateway daemon runs on one machine that provides the Ω MEGA service. It can start the actual Ω MEGA process and its the associated modules on any of the servers, depending on their current work load. In this way, we are able to employ the whole computational power of a local area network with a background of several larger servers.

But before we discuss issues of further distribution let us take a closer look at possible mathematical services.

3 Mathematical Services

An attempt at a categorization of mathematical services has been made in [HC96a]. As possible services the authors have concentrated on systems for reasoning, i.e. theorem provers, and systems for computation, i.e. Computer Algebra Systems. Here we both extend the list of services and furthermore group them more thoroughly. Moreover we will be less formal in defining the different categories of systems by giving a more intuitive description of their working principle. Yet in general every mathematical service mentioned here can be viewed as a combination of an implementation of a mathematical computation or processing together with an interaction component that controls the communication with other available services.

3.1 Mathematical Filters

Certain mathematical programs can be used in a filter-like way, that is they can read a request from an input stream and write some answer to an output stream. Mathematical filters can be further

grouped in **computation filters** and **deduction filters**. The first perform some numerical or algebraic calculation which result they return (maybe coupled with some protocol information on how the result was obtained), while the latter attempt to prove a given problem and return, if successful, the proof or signal failure. Unlike computation filters which terminate eventually, deduction filters will not always return a result. Thus deduction services need to have additional properties for maintenance: On the one side a requesting client must be able to send a termination signal to a deduction service in order to declare an earlier request as obsolete. On the other side the service itself needs to survey its own running processes, assign resources to incoming requests and terminate processes that have not produced any results after their allocated resources have been consumed.

An Example: SPASS In an experimental implementation we have established several service-points for automated theorem provers. One of these is for the prover SPASS [Wei97], that usually communicates via an input and an output file. SPASS can now be accessed by sending a string to its service-point that is converted into the required input file. Conversely the service-point returns the SPASS output file to the requesting client in string format. In order to guarantee termination the client can send a kill signal to end its own requested processes and furthermore the SPASS service-point terminates processes after a certain time limit has expired, which adjusts itself dynamically according to the number of incoming requests.

3.2 Knowledge Bases

Contrary to other mathematical services knowledge bases have the property that they can be dynamically changed by some clients. As trading points only permit or deny access to whole services some additional information on access rights to knowledge bases have to be maintained by respective service points they have enrolled. They can then allocate permission to customers whether they are allowed to only retrieve knowledge or are also enabled insert and possibly delete facts.

An Example: The Mizar Library The Mizar Library [Rud92] is a knowledge base that already offers its services via the Internet. It has been successfully put to use in an integration of Mathematica [Wol96] and ILF [Dah97], where theorems can be proved from within an ILF-notebook in Mathematica by using ILF's ability to employ automatic theorem provers with facts retrieved from the Mizar Library.

3.3 Control Components

Control components process data received from clients by editing and distributing it to other mathematical services. Thereby data can be translated either into the special input language of a designated service or a syntax common to all available services. Thus control components can be viewed as front ends to mathematical services that simplify the access for both human users and other systems.

3.4 Display Components

This point covers possible output devices that enable a user to view processed mathematical data in a desired way. To these services belong (graphical or non-graphical) displays and browsers for formulas and proofs, as well as systems that can transform provided data into a certain output format. As an example of the latter, one might consider systems that translate proofs into natural language.

An Example: $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ The graphical user interface $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ combines both control and display component for the Ω MEGA system. It adapts its menu structure according to the available services (i.e. which external reasoners such as ATP or CAS can be used) and to the mathematical knowledge

that has been acquired from a knowledge base. It furthermore coordinates the communication between the core of the Ω MEGA system and other mathematical services.

3.5 Anytime Services

Anytime services provide a mean to organize the output of computations that might have more than one result (possibly infinitely many results) to a clients request. The general functionality of these services is similar to those of mathematical filters, but they are moreover able to store additional information on both the requested service and the costumer. The latter itself receives a result along with informations on how long the anytime service can provide further results and how these results can be retrieved. Using these specifications, subsequent requests of the client can then be answered by the anytime service using the already computed results (always provided the requests are within the given time limit).

Some anytime services might have properties similar to these of deduction filters, i.e. they might not terminate. Therefore they have to be administered by the responsible service-points accordingly.

An Example: Unification engine for higher order logics As higher order unification is infinitary, such algorithms usually have the property to provide incrementally more unifiers for a given problem. Clients in need of a solution to a higher order unification problem can send a complying request to the service. The unification engine then returns the first possible unifier (if one exists) while keeping the information on the problem. If the client is not satisfied with the first solution it can demand more unifiers (if they exist), which the unification service computes successively one at a time using the background of the old computations.

3.6 Mathematical Servers

Finally mathematical servers form the link between several different other mathematical services, by integrating them as permanent or temporary components of their own working environment. Furthermore they are the only services control components can connect to as clients permanently.

They again can be divided into **computation** and **deduction servers**, yet this distinction does not have any impact on their behavior but simply states their primary purpose. Thus deduction servers are used mainly for the task of proving theorems whereas the main purpose of computation servers is their use for algebraic computations and development of algorithms for symbolic calculations.

An Example: The Ω MEGA Service The Ω MEGA Service forms the very core of the Ω MEGA System. It contains the basic functionality for the reasoning process, such as the structures for terms, proofs, etc. It can be accessed by permanently connecting with the help of a control component like $\mathcal{L}\Omega\mathcal{M}\mathcal{I}$. $\mathcal{L}\Omega\mathcal{M}\mathcal{I}$ then acts as a client to the Ω MEGA service and coordinates contacts from Ω MEGA to other services.

4 Towards a Deduction Meta-Language

In the current implementation of $Oz-DMS$, a proprietary **protocol** is run between the mathematical services that is customized to the current needs and functionality of Ω MEGA. For example, requests for proofs and the partial proofs themselves are communicated in the Ω internal format. On the other hand, spawned services can be interrupted by a special message. In order to move towards an open architecture that is not necessarily restricted to particular applications and modules and that allows for a variety of topologies, this is likely to change into a communication **language**.

In intelligent agent research, a reasonable, nested distinction of communication languages into *interlingua* and *ontolingua* has been proposed. *Ontolingua*, such as the Knowledge Interchange

Format (KIF) [Gea92], standardize the actual objects of communication. In the case of mathematical theorem proving, these objects are theorems, theories, but also (partial) proofs, and even proof plans. Appropriate candidates for such a language are the so-called DFG syntax [HKW96] or, if integrating symbolic computation services, the protocol that is put forward by the OpenMath initiative [AvLS96], which strives for a standard CAS communication protocol.

While ontolingua are designed for transporting representations of a particular domain or application, software agents, such as our mathematical services, indeed have to negotiate about these representations, e.g., requesting the proof of a lemma from a service/from the user or being told its actual derivation afterwards. These *performatives* or *speech acts* defined by means of so-called *interlingua* are certainly domain-independent in that they build a ‘context’ wrapped around the actual content of a message encoded in the object language. The most prominent and complete approach is the KQML [FF94] specification. It consists of about twenty performatives such as request, tell, reject, etc., that handle a whole variety of standard situations including those of service-providing agents.

Once equipped with such standardized communication facilities, it is not only possible to connect mathematical services to the rest of the networked world. The richness of interaction between modules working in a particular theorem proving environment allows for a far more flexible and decentral flow of control.

The contract net protocol [Smi80] and its derivatives, for example, can be expressed by means of KQML performatives and install a simple, but powerful market mechanism to optimize the workload of a system without requiring any central decision maker. An agent herein ‘announces’ a task, such as the proof of a certain theorem, to a number of other service agents. Each service now judges its competence and predicts the expected costs that his processing will produce. It ‘bids’ for the task accordingly. The initiative agent then selects one or several service agents in order to reduce costs and maximize performance.

A further advantage of such an open approach is that several users with different demands can use the system cooperatively or independently at the same time. The particular modules then decide based on priority and workload whether to process particular tasks or not.

5 Related Work

In [FI98], Fischer and Ireland propose an agent-based approach to proof planning that is however mainly motivated by a fine-grained parallelization of the proof planning process more than the distribution aspect. They propose a society of agents that are organized by a contract net architecture, building on earlier studies of Fisher [Fis97] on agent-based theorem proving.

Calmet and Homann present a framework for establishing the semantics of intimately integrated deduction and computation systems [Hom96, HC96b]. In a servicing architecture like the one described in this paper, the semantics of the protocol employed in the communication is not a correctness problem, since our approach assumes that proofs are communicated, so that the initiator of a reasoning task can always collect the partial proofs and verify the correctness of the final resulting proof if he does not trust the mathematical services.

To our knowledge, only three distributed theorem proving systems besides Ω have actually been implemented up to now. The modal-logic theorem prover from [Pit96] uses a trader model like the one realized in Ω MEGA, the ILF system [Dah97] connects to MATHEMATICA and some automated theorem provers in a simple master-slave model and a group of experimental systems centering around DISCOUNT theorem prover [DKS97, DF97]. The last experiments explore the aspect of a tight cooperation between the theorem provers that makes a group mathematical services more successful than any single component.

6 Conclusion

We have presented a distributed network architecture for automated and interactive theorem proving that supports modularization, distribution, inter-operability, networking, and coordination of mathematical services. We have described an implementation, *Oz-DMS*, which provides the functionality to turn existing theorem proving systems and tools into mathematical services that are homogeneously integrated into a networked proof development environment.

Acknowledgements

The work reported here was supported by the “Deutsche Forschungsgemeinschaft” (DFG) in Sonderforschungsbereich 378 and Graduiertenkolleg “Kognition”.

The authors would like to thank the Ω MEGA group at the Universität des Saarlandes for valuable discussions Furthermore, we would like thank the Oz development team at the Universit”at des Saarlandes. Without the the Oz programming language and their readily shared know-how, the whole enterprise of distributing Ω MEGA and developing *Oz-DMS* would have been impossible.

References

- [ABI⁺96] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.
- [AvLS96] J. Abbot, A. van Leeuwen, and A. Strotmann. Objectives of OpenMath. technical report 12, RIACA, Technische Universiteit Eindhoven, The Netherlands, June1996.
- [BCF⁺97] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. Ω MEGA: Towards a mathematical assistant. In McCune [McC97], pages 252–255.
- [BF94] Peter Baumgartner and Uli Furbach. PROTEIN, a PROver with a Theory INterface. In Bundy [Bun94], pages 769–773.
- [BHC95] C. Ballarin, K. Homann, and J. Calmet. Theorems and algorithms: An interface between isabelle and maple. In A. H. M. Levelt, editor, *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC’95)*, pages 150–157. ACM Press, 1995.
- [BK98] Christoph Benzmüller and Michael Kohlhase. LEO, a higher-order theorem prover. to appear at CADE-15, 1998.
- [Bun94] Alan Bundy, editor. *Proceedings of the 12th Conference on Automated Deduction*, number 814 in LNAI, Nancy, France, 1994. Springer Verlag.
- [CP98] J. Cannon and C. Playoust. *Algebraic Programming with Magma*, Volume1, 2. Springer-Verlag, 1998. forthcoming 2/98.
- [Dah97] Ingo Dahn. Integration of automated and interactive theorem proving in ILF. In McCune [McC97], pages 57–60.
- [Den93] Jörg Denzinger. *Teamwork: A method to design distributed knowledge based theorem provers*. PhD thesis, Universität Kaiserslautern, 1993. (in german).
- [DF97] Jörg Denzinger Dirk Fuchs. Knowledge-based cooperation between theorem provers by techs. Seki Report SR-97-11, Fachbereich Informatik, Universität Kaiserslautern, 1997.

- [DKS97] Jörg Denzinger, J. Kronenburg, and Stephan Schulz. Discount - a distributed and learning equational prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.
- [FF94] T. Finin and R. Fritzson. KQML — a language and protocol for knowledge and information exchange. In *Proceedings of the 13th Intl. Distributed Artificial Intelligence Workshop*, pages 127–136, Seattle, WA, USA, 1994.
- [FI98] Michael Fisher and Andrew Ireland. Multi-agent proof-planning. to appear at the CADE-15 Workshop “Using AI methods in Deduction”, 1998.
- [Fis97] Michael Fisher. An open approach to concurrent theorem proving. In J. Geller, H. Kitano, and C. Suttner, editors, *Parallel Processing for Artificial Intelligence*, Volume 3. Elsevier/North Holland, 1997.
- [Gea92] M. Genesereth and R. Fikes et al. Knowledge interchange format: Version 3.0 reference manual. technical report, Computer Science Department, Stanford University, 1992.
- [HC96a] K. Homann and J. Calmet. Structures for Symbolic Mathematical Reasoning and Computation. In Jaques Calmet and Carla Limongelli, editors, *Design and Implementation of Symbolic Computation Systems; International Symposium, DISCO '96; Proceedings*, Volume 1128 of LNCS, pages 217–228, Karlsruhe, Germany, September 18–20 1996. Springer Verlag, Berlin, Germany.
- [HC96b] Karsten Homann and Jacques Calmet. Structures for symbolic mathematical reasoning and computation. In Jacques Calmet and Carla Limogelli, editors, *Design and Implementation of Symbolic Computation Systems, DISCO'96*, number 1128 in LNCS, pages 216–227, Karlsruhe, Germany, 1996. Springer Verlag.
- [HKK⁺94] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Daniel Nesmith, Jörn Richts, and Jörg Siekmann. Ω -MKRP a proof development environment. In Bundy [Bun94], pages 788–792.
- [HKW96] Reiner Hähnle, Manfred Kerber, and Christoph Weidenbach. Common syntax of dfgschwerpunktprogramm “deduktion”. Interner Bericht 10/96, Universität Karlsruhe, Fakultät für Informatik, 1996.
- [HLS⁺96] Dieter Hutter, Bruno Langenstein, Claus Sengler, Jörg H. Siekmann, Werner Stephan, and Andreas Wolpers. Verification support environment. *High Integrity Systems*, 1(6), 1996.
- [Hom96] Karsten Homann. *Symbolisches Lösen mathematischer Probleme durch Kooperation algorithmischer und logischer Systeme*. PhD thesis, Unversität Karlsruhe, 1996.
- [HT93a] J. Harrison and L. Théry. Extending the HOL Theorem Prover with a Computer Algebra System to Reason About the Reals. In C.-J. H. Seger J. J. Joyce, editor, *Higher Order Logic Theorem Proving and its Applications (HUG '93)*, Volume 780 of LNCS, pages 174–184. Springer Verlag, Berlin, 1993.
- [HT93b] John Harrison and Laurent Théry. Reasoning About the Reals: The Marriage of HOL and Maple. In A. Voronkov, editor, *Proceedings of the 4th International Conference on Logic Programming and Automated Reasoning (LPAR'93)*, Volume 698 of LNAI, pages 351–353, St. Petersburg, Russia, Juli 1993. Springer Verlag, Berlin.
- [KKS98] Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating computer algebra into proof planning. *Journal of Automated Reasoning*, 1998. Special Issue on the Integration of Computer Algebra and Automated Deduction; forthcoming.
- [McC97] William McCune, editor. *Proceedings of the 14th Conference on Automated Deduction*, number 1249 in LNAI, Townsville, Australia, 1997. Springer Verlag.

- [MW97] William McCune and Larry Vos. Otter CADE-13 competition incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997. Special Issue on the CADE-13 Automated Theorem Proving System Competition.
- [Pit96] Jeremy Pitt. A WWW interface to a theorem prover for modal logic. In Nicholas A. Merriam, editor, *User Interfaces for Theorem Provers*, pages 83–90, Department of Computer Science, University of York, UK, 1996.
- [Rud92] Piotr Rudnicki. An overview of the mizar project. In *Proceedings of the 1992 Workshop on Types and Proofs as Programs*, pages 311–332, 1992.
- [Smi80] R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *IEEE Transaction on Computers*, number 12 in C-29, pages 1104–1113, 1980.
- [Smo95] G. Smolka. The Oz Programming Model. In Jan van Leeuwen, editor, *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pages 324–343. Springer-Verlag, Berlin, 1995.
- [Wei97] Christoph Weidenbach. SPASS: Version 0.49. *Journal of Automated Reasoning*, 18(2):247–252, 1997. Special Issue on the CADE-13 Automated Theorem Proving System Competition.
- [WJ95] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.
- [Wol96] Stephen Wolfram. *The Mathematica book : version 3.0*. Wolfram Media, Inc., Champaign, IL, USA, 3rd edition, 1996.