

Deliverable 4.1:
Survey of Existing Tools for Formal MKM

Mathematical Knowledge Management Network
MKMnet, IST-2001-37057

Contents

1	Languages for representing formal mathematics	5
1.1	MATHML	5
1.2	OPENMATH	5
1.3	OMDOC	6
1.4	FoC	6
1.5	MIZAR	7
1.6	Others	7
2	Existing repositories/libraries of formalized mathematical knowledge	8
2.1	MBase	8
2.2	MIZAR	8
2.3	Conclusions	10
3	Current techniques and tools for organizing and presenting mathematical knowledge	11
3.1	ActiveMath	11
3.2	Automath	11
3.3	MathLang	12
3.4	MIZAR	12
3.5	Theorema	13
3.6	Conclusions	13
4	Algebraic Specification methods and tools	15
4.1	Motivation and historical remarks	15
4.2	CASL	15
4.3	The CASL Tools Set (CATS)	15
4.4	Other Tools and Methods	16
4.5	Assessment and Conclusions	16
5	Automated Reasoning Systems	18
5.1	First-Order Theorem Provers	18
5.2	Inductive Theorem Provers	20
5.3	Higher-Order and Type Theory Proof Systems	22
5.4	Human Computer Interfaces to Theorem Provers	27
6	Computer Algebra Systems	30
6.1	Introduction	30
6.2	Mathematica	31
6.3	Maple	32
6.4	Reduce	32
6.5	MuPAD	33
6.6	Other Systems	34
6.7	Conclusions	35

7	More general mechanisms for representing resources	36
7.1	HELM	36
7.2	Mowgli	38

Summary

The present document is the result of **Task 4.1: Survey of current techniques and tools and assessment of their usability for MKM**, of the **Work Package 4: Formal Tools in MKM**.

The purpose of this task is to share the present expertise and the different points of view of the various groups.

The sections have been compiled by various partners in the project, using information from other groups when necessary:

1. Languages for representing formal mathematics *UPMC Paris 6*: Renaud Rioboo.
2. Existing repositories/libraries of formalized mathematical knowledge: *University of Saarlandes*: Christoph Benzmueller.
3. Current techniques and tools for organizing and presenting mathematical knowledge: *Heriot Watt*: Fairouz Kamaredine.
4. Algebraic specification methods and tools: *SCC Hagenberg*: Franz Lichtenberger
5. Automated reasoning systems: *DFKI Saarlandes*: Dietter Hutter.
6. Computer Algebra systems: *RISC-Linz*: Tudor Jebelean.
7. More general mechanisms for representing resources *University of Bologna*: Andreea Asperti.

The complete report has been collected and integrated by Tudor Jebelean, RISC-Linz.

1 Languages for representing formal mathematics

“Representing formal mathematics” is an expression whose meaning must be clarified in MKM’s context. Any theorem prover enables to represent “formal mathematics” and should thus be listed in this section. Since those will be addressed elsewhere in this document, we will not discuss them here.

Similarly, any computer algebra system embeds a way to manipulate formulas and thus qualifies as a tool for representing “formal mathematics”. Again, these will be listed elsewhere and will be ignored in this section.

In this subtask we will thus understand “representing formal mathematics” in a very general sense: a representation which is suitable for computer processing, but it does not assume the use of a particular theorem prover or computer algebra system. As an exception from this rule we will however present the MIZAR language - which is somehow at the border (it is treated by a special proof-checker, but it is intended as a general language).

1.1 MATHML

MATHML is a W3C ([http\string://www.w3c.org/](http://www.w3c.org/)) recommendation which enables to both transmit and display mathematical formulas. In its current form, MATHML understands a fixed set of mathematical objects which covers “mathematics for the undergraduate student”. Fully specified using W3C’s XML specifications, MATHML has become the de-facto standard for distributing mathematical formulas across the WEB. It can be understood as a simplification of $\text{T}_{\text{E}}\text{X}$ or $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ macros used inside scientific documents to display mathematics.

MATHML’s strength is that the layout of formulas is not part of the document but is delegated to a browser through “recommendations” of the standard. It is thus extremely portable and, high quality, renderers of MATHML documents are now widely available. AMAYA W3C’s browser was the first to display MATHML documents. Today, renderers such as Mozilla or Netscape-7 in the Unix world or IBM’s TECHEXPLORER plugin for Internet Explorer offer rendering qualities which are close to $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ produced documents.

Being fully specified in XML, MATHML benefits of universally available tools to manipulate XML documents. In particular, the extended stylesheet language XSL often enables to quickly design an interface between the specific format of a computer algebra system or a theorem prover and MATHML. Nowadays most major computer algebra systems such as MATHEMATICA or MAPLE are MATHML aware for reading or displaying mathematical formulas.

1.2 OPENMATH

MATHML’s format is not extensible, and higher mathematics can hardly be described using it. MATHML’s recommendations focus on the display of formulas and not on their meaning. OPENMATH’s framework ([http\string://www.openmath.org/](http://www.openmath.org/)) extends MATHML capabilities by attaching a meaning to a particular symbol. In order to be extensible, OPENMATH’s symbols are defined by “content-dictionaries”, which are “coherent sets” of symbols. A content-dictionary thus defines a set of symbols together with an informal

description of the meaning, characteristic properties and sample uses of each symbol it defines.

A particular content-dictionary `mathml` enables to view MATHML as a strict subset of OPENMATH. Let us imagine that two applications want to exchange or share mathematical objects. Using MATHML, they would need to be compliant with its fixed set of symbols and recommendations. Using OPENMATH, they need to first agree on a common set of content-dictionaries (called a CD group) prior to exchanging or sharing mathematical data.

Content-dictionaries thus add expressiveness to the language. Beyond OPENMATH's standard content-dictionaries, users may define new dictionaries for their particular applications. They enable to provide an informal "meaning" to "words" of a mathematical "sentence". OPENMATH is thus an extension of MATHML which enables user-defined symbols through the mechanism of content-dictionaries.

1.3 OMDoc

Though OPENMATH reliably describes formulas, it can hardly describe their meaning in a given mathematical context. Words of mathematical sentences can have different meanings in different branches of mathematics. The purpose of the OMDOC standard is to enhance OPENMATH with capabilities to describe the mathematical context where OPENMATH's objects are used.

OPENMATH is a replacement for MATHML technology. On the contrary, OMDOC (<http://www.omdoc.org/>) enhances OPENMATH in the sense that it uses OPENMATH to describe formal mathematical objects. It can be viewed as an application of OPENMATH which extends its capabilities by enabling the mathematician to provide informal descriptions of *axioms*, *lemmas*, *examples* or *proofs*. Like L^AT_EX tends to replace a mathematician's pen, OMDOC aims to offer him a structured and semi formalized way to write mathematics. In short, OPENMATH describes formulas of an OMDOC book.

OMDOC is the emerging standard for applications which make advanced use of mathematical objects. An interesting application of OMDOC is the MATHWEB (see more on <http://www.mathweb.org/>) framework which enables different applications to exchange mathematical "knowledge" through a "MATHWEB Software Bus". Applications connected on this bus include the MBASE database repository of OMDOC mathematical documents. Another application using the MATHWEB framework is the ACTIVEMATH WEB-based interactive tools for learning mathematics.

1.4 FOC

In previous sections we have used the words "informal" or "semi formal" in the sense that coherence is not automatically verified. It is the responsibility of the writer of an OPENMATH content-dictionary to ensure that it does not conflict with another one. Similarly, the writer of an OMDOC book must ensure coherence when including parts of several OMDOC documents.

When one reuses parts of existing OPENMATH content-dictionaries or of OMDOC books, one must ensure that no conflicts occur. This can be very painful, particularly when having in mind that data formats are not human oriented.

The purpose of the FOC system is to provide a framework to describe and compute with mathematical objects. It aims at providing a certified computer algebra library.

A recent work, by Manuel Maarek at Paris 6, has extended the FOC system with descriptive capabilities. This would ideally translate a FOC program into an OMDoc book, thus enabling a comprehensive interpretation of the program. Since FOC's purposes are coherence and computations, sources are deeply analyzed and verified by the FOC compiler. These analysis enable to translated the constructive part of the program into an execution language (OCAML). The descriptive part of the code is verified by a proof assistant (COQ) providing mechanized proofs of the program.

Translating an OMDoc book into FOC specifications is also possible and would enable to check its consistency.

1.5 MIZAR

The MIZAR language had been designed by Andrzej Trybulec to be as close as possible to the mathematical vernacular and, simultaneously, simple enough to allow the texts written in it to be processed by computer.

MIZAR includes the reconstruction of the core of mathematics. Classical logic in Fitch–Jaśkowski style is a logical framework of it. The original goal of Jaśkowski was to reconstruct the system of logic used in mathematical papers as opposite to the system described by logicians. Typical use of MIZAR, namely Mizar Mathematical Library (MML), is based on Tarski Grothendieck set theory (TG). TG is slightly stronger system than commonly accepted by mathematicians (ZFC). But it has negligible impact on the development of MML. Namely, it is used in mathematical papers related to advanced category theory.

The syntax of MIZAR is sufficiently reach to write mathematical texts in comfortable way. Particularly special attempt had been made to ensure proper semantics of such an important part of mathematical language as the system of adjectives. MIZAR allows dependent types which are frequently used in regular mathematics.

The characteristic feature of MIZAR language is its linguistic superstructure. The core of the MIZAR language (so called "strict Mizar") has the same expressive power like whole MIZAR but is a rather simple language and can be easily mastered by beginners. The whole MIZAR is sufficiently reach to allow writing papers in a convenient way by more experienced authors.

1.6 Others

We have seen that beyond "presentation", formalized mathematics need "representations" which can be automatically analyzed. A theorem prover checks consistency of mathematical statements. Any processing of the proof developed in the system is a form of manipulation of formalized mathematics. In particular, attention must be paid on techniques which display a proof in a human readable form. The THEOREMA proof printer is such an example. Another example of manipulation of formal mathematical parts of a proof is the CENTAURE environment developed to facilitate the development of COQ programs.

2 Existing repositories/libraries of formalized mathematical knowledge

2.1 MBase

MBase is a knowledge management tool for mathematical content encoded in OMDoc format. OMDoc is a description language tailored for purposes appearing in symbolic sciences as logics, mathematics computer science etc. It distinguishes between informal and formal content such that communication with automated services as theorem provers, computer algebra systems or databases is feasible. OMDoc comes as a specialized XML-format. Though MBase stores the information within a relational (SQL) database for (search) efficiency. MBase may be tailored to use extensions of OMDoc as long as these are not in conflict with OMDoc notions. In practice this is already done to import ActiveMath content. MBase distinguishes several categories of information objects, on which the structure of the underlying data base model is grounded: Definitions for associating new symbols with terms from already introduced ones, Assertions being the counterparts of axioms, theorems, conjectures and lemmas with or without associated proofs, Examples are extra for obvious practical purposes, Theories group mathematical objects as the ones aforementioned and knowledge for construction of inheritances between theories, Metadata in form of Dublin Core definition are supported plus a possibility of additions not matching that standard. Requests from automated services are supported via standardized interfaces (XMLRPC protocol). Further information on MBASE is available in published papers¹.

2.2 MIZAR

The information objects of MIZAR are fewer than in OMDoc - MIZAR mainly builds on Theorems, Definitions and Schemes (theorems with second order variables). This is due to easy readability in the traditional sense of mathematicians and for easier controllable programming. Its strength lies in handling proof notions and relating mathematical descriptions (more than 90 keywords in the full MIZAR version). A "Strict Mizar" with the same expressibility but a smaller core is on a verge of completion (mainly for didactic purposes). Thus the main notions of OMDoc and MIZAR can be seen as rather good related. Further information on MIZAR is available at www.mizar.org.

The data base model of MBase contains some relations between objects of these kinds onto which inheritance is made: Definition entailment for relating defined symbols to others, Depends-on/Local-in for specifying dependency and locality information for primary knowledge items (the use of symbols in a definition or assertion is done explicitly as well as for applications of lemmas in proofs), Theory inheritance.

Mizar Mathematical Library (MML) is a repository of articles written in the MIZAR language. It consists of 743 articles (November 1, 2002) authored by 135 authors from 10 countries (Poland 88, Japan 28, Canada 7, Germany 4, China 2, Russia 2, Czech Republic

¹A. Franke, M. Kohlhase. System Description: MBASE, an Open Mathematical Knowledge Base. CADE'00, pp. 455–459.

M. Kohlhase, A. Franke. MBase: Representing Knowledge and Context for the Integration of Mathematical Software Systems. *J. of Symbolic Computation*, vol. 32, no. 2, Sep. 2001, pp. 365–402.

1, the Netherlands 1, Spain 1, USA 1). The policy is liberal and virtually all submitted articles are accepted, under conditions they are MIZAR correct, i.e. accepted by the MIZAR verifier. However, quite often the Library Committee of SUM² helps the author to enhance the submitted article e.g. in extreme case to submit one short article instead of three long ones.

The MIZAR language (see also Subsection 1.5) is intended to represent mathematical knowledge in the manner to which working mathematicians are accustomed.

MML is revised frequently. Revisions include enhancement of the proofs, generalizing definitions and theorems, reorganizing the order of processing of articles. Revisions are made using automatic tools or by hand. The most important implemented automatic tools for revisions are the following: RELPREM that removes unnecessary references; RELINFER that removes unnecessary steps in proofs; RELITERS that removes unnecessary steps in calculations; TRIVDEMO that discovers trivial proofs, i.e., proofs that maybe replaced by straightforward justifications.

As a rule the new articles are immediately revised.

MML includes well developed theory of basic mathematical notions introduced in set theory and arithmetic.

The quantitative data of MML:

– number of articles:	743
– size of MML in bytes:	55,968,658
– number of theorems:	32,752
– number of definitions:	6,263
– number of registrations:	4,926
– number of external references:	391,463

Among others MML includes the following theorems:

Abian Fixpoint Theorem, Alexander Lemma, Baire Category Theorem, Birkhoff Theorem of Varieties of Manysorted Algebras, Fixpoint Theorem for compact spaces, Fundamental Theorem of Algebra, Hahn Banach Theorem, Hessenberg Theorem, Jónson Theorem of representation of lattices, König Lemma, König Theorem, Newman Lemma, Reflection Theorem, Steinitz Theorem, Stone Theorem: Paracompactness of metrizable spaces, Tarski Knaster Fixpoint Theorem, Urysohn Lemma.

In last few years the main efforts are concentrated on

- formalizing the handbook *A Compendium of Continuous Lattices*, G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. Springer-Verlag, Berlin, Heidelberg, New York, 1980.
- proving the Jordan Curve Theorem
- developing a theory of Random Access Turing Machines

²Association of Mizar Users

Encyclopedia of Mathematics in Mizar The work on proper organization of knowledge stored in MML has just started. It is the work on the so called Encyclopedia of Mathematics in Mizar (EMM). The distinction between MML and EMM corresponds strictly to the difference between primary and secondary knowledge in the meaning used in Library and Information Sciences. As a case study we prepared new *encyclopedia* articles XBOOLE_0 and XBOOLE_1. XBOOLE_0 had been prepared after careful discussion. Then, an automatic tool, namely MML QUERY, was used to extract theorems, schemes and registrations from MML that use only the conceptual framework provided by XBOOLE_0.

Formalized Mathematics Formalized Mathematics is a journal (ISSN 0777-4028, 1426-2630) that publishes the contents of MIZAR articles submitted to MML. It was established in 1990. Up to now there were 28 issues published in 9 volumes. The publication is obtained by automatic translation from MIZAR language into English with mathematical symbolism available in T_EX. The tools for automatic translation are still being developed.

The electronic version of the journal is available at `://mizar.uwb.edu.pl/JFM/`.

2.3 Conclusions

With respect to Task 4.1 MBASE and MIZAR may be used

1. for (distributed) repositories for mathematical knowledge which are used by an author for storing and retrieving publications; one is not restricted to a single MBASE. MIZAR may be used for proving correctness and for developing a convenient formalization of the (parts of) a document in process.

A task to follow is MIZAR to OMDOC conversion (abstracts and full text) of MIZAR articles (collected in the Journal of Formalized Mathematics). MBASE may also help in referencing or finding related or needed objects. MML QUERY provides semantic search for MML with well developed system of queries. Still there is a room for further improvements. Particularly, automatic structuring given by MBASE may be used for this purpose. MBASE enables also sharing experiences between different systems.

2. for a unified representation of different texts - this means presentation and code (only partially overlap with point 1); there is already a possibility of translating TPS, Omega and PVS content to OMDoc documents. Thus an MBASE may help for collecting content (sometimes checked or proved) from work performed with different math-assistants. One has to observe that MIZAR is based on classical first-order logic and Tarski-Grothendieck set theory. OMDoc supports relations between theories and especially different logical foundations. MIZAR articles belong to the classical logics foundation. An interesting approach could be a comparison to non-classical proof systems.

Systematizing content is possible in this way, yet MBASE supports such features only roughly due to lack of implemented features from the specification. An ongoing development is a full exchange (with proofs) between MBASE and other systems (especially between MIZAR and MBASE).

3 Current techniques and tools for organizing and presenting mathematical knowledge

3.1 ActiveMath

ActiveMath is a learning environment for mathematics developed at the DFKI in Saarbruecken and at the university of Saarbruecken. See <http://www.activemath.org/> where nice demos are also available.

The goal of the project's research and development is a web-based interactive learning system (for mathematics) that uses instruction as well as constructivist elements. The project provides an architecture, basic knowledge representations, and techniques for new-generation on-line interactive mathematics documents (textbooks, courses, tutorials) and e-learning.

ActiveMath supports amongst other things, the reusability of the encoded content, individualized learning with a user-adaptive environment, and active learning by using external tools enabling exploratory learning.

ActiveMath techniques were developed and its prototype systems were tested with interactive textbooks on Algebra, Analysis statistics and optimization.

3.2 Automath

The Automath project ³started in 1967 by N.G. de Bruijn who wanted it to be *not just meant as a technical system for verification of mathematical texts, it was rather a life style with its attitudes towards understanding, developing and teaching mathematics*. De Bruijn added: *The way mathematical material is to be presented to the system should correspond to the usual way we write mathematics. The only things to be added should be details that are usually omitted in standard mathematics*. De Bruijn spent a lot of effort on this goal and studied the language of mathematics in great depth. Together with his PhD student Bert van Benthem-Jutting they built an impressive student which was used to fully check the whole book of Landau on foundations of analysis. The system remains one of the most influential to date in the field of theorem proving and proof checking. Reading the literature of the proof checker Nuprl of Cornell, USA, the theorem prover Coq of France, and the Logical Frameworks of Edinburgh, it becomes evident that the influence of Automath on these systems is beyond question.

In his languages for Automath, de Bruijn incorporated techniques to represent the 'administrative structure' of mathematical texts where one can account for contexts, parameter lists and variables. Two distinctive feature of Automath are as follows:

- The use of books. That is, like a mathematical text, Automath is written line by line, where each line refers to definitions or results given in earlier lines.
- The use of definitions. Without definitions, expressions very soon become too long. Moreover, a definition gives a name to a certain expression, and this name makes it easier for the user to remember (or understand) what the use of the definiens is.

³R. P. Nederpelt, J. H. Geuvers, R. C. de Vrijer. *Selected Papers on Automath*. North-Holland, Amsterdam, 1994.

3.3 MathLang

De Bruijn presented his Mathematical Vernacular MV in two rounds (cf. F3 of ⁴,Section 1.11, p. 868): *In the first round we express the general framework of organization of mathematical texts. It is about books and lines, introduction of variables, assumptions, definitions, axioms and theorems... In the second round we get the rules about validity.* But MV has more logic incorporated in it than is found in mathematical texts. The *mathematical vernacular* imposes logical choices and correctness conditions. Furthermore, Automath, requires full formalization.

To avoid these disadvantages, a language MathLang based on de Bruijn's MV and his system Automath is being developed in order to better meet the demand of mathematicians and users of mathematics. This language starts from a calculus which is close to MV ⁵redesigning it as necessary for the full translation of full mathematical texts. We are translating the Landau book in MathLang to compare it with the formalisation in Automath. The basic idea of this style is that none of the logical languages of the 20th century satisfies the criteria expected of a language of mathematics. A logical language does not have mathematico-linguistic categories, is not universal to all users of mathematics, and is not a satisfactory communication medium. Moreover:

- Logical languages make fixed choices (first versus higher order, predicative versus impredicative, constructive versus classical, types or sets, etc.). But different parts of mathematics need different choices and there is no universal agreement as to which is the best formalism.
- A logician writes in logic their understanding of a mathematical-text as a formal, complete text which is structured wildly unlike the original, and is of little use to the *ordinary* mathematician.
- Mathematicians do not want to use formal logic and have for centuries done mathematics without it.

3.4 MIZAR

The MIZAR system is implemented in Free Pascal and available on the platforms: Linux, Windows, Solaris x86.

The MIZAR script, so called MIZAR article, can be prepared using any ASCII editor. Then, it can be checked by MIZAR VERIFIER that discovers and reports errors. Errors must be corrected by hand and then MIZAR VERIFIER is called again. So, a lazy interaction is used to develop an article. Most often it is done by stepwise refinement: the author writes a proof plan and afterwards fills the (logical) gaps reported. For the emacs editor the MIZAR mode is implemented that facilitates writing the articles. An author may also use MML QUERY – the search engine for MML.

⁴R. P. Nederpelt, J. H. Geuvers, R. C. de Vrijer. *Selected Papers on Automath*. North-Holland, Amsterdam, 1994.

⁵F. Kamareddine, R. P. Nederpelt. A refinement of de Bruijn's formal language of mathematics. *J. Logic, Language and Information*, 2003, in print.

The system is available on WWW at <http://mizar.uwb.edu.pl/> and <http://mizar.org/>.

The distribution of the MIZAR system and the Mizar Mathematical Library are free of charge for non commercial purposes.

3.5 Theorema

The goal of the Theorema project (www.theorema.org) is to provide computer support for all phases of the mathematical development cycle: to prove theorems, use them for computations and experiments, conjecture new theorems, prove those, extract solution methods from them, apply them in other proofs, etc. The *Theorema* system, which is being developed in the frame of this project, offers these features in one coherent software system which is currently based on the rewrite engine of the computer algebra system *Mathematica*.

The particular features of the system include:

- a simple and intuitive set of commands for writing and using mathematical knowledge (MK);
- the use of higher-order predicate logic as the natural language for expressing MK;
- a collection of provers for general logic as well as for specific domains.

Theorema represents mathematical knowledge in formulae expressed in the language of higher-order predicate logic. The input and output of the formulae is done in natural 2D notations, very similar to the one used by working mathematicians. This is possible because the underlying *Mathematica* software, which provides a rich set of special symbols, as well as the possibility of defining the external appearance of expressions in the user interface. (The latest release also allows user-defined symbols, which in Theorema are used for introducing new “logico-graphic” symbols.)

For defining MK, Theorema provides to the user commands like **Definition**(*name*, *formula*) (also **Lemma**, **Axiom**, etc.), which stores the formula[s] into the system, associated with the respective name and possibly with certain user defined labels. After the formula is stored into the system, it can be referenced later as **Definition**(*name*) (respectively **Lemma**(*name*), etc.). For structuring MK, Theorema provides the command **Theory**(*name*, *list*), which associates *name* with a list of formulae and/or other theories. The theory can be referenced later as **Theory**(*name*), and can be also included in other theories. This allows for structuring arbitrarily complex theories with sub-theories, etc.

For using MK, Theorema provides three basic commands: **Prove**, **Compute**, and **Solve**. An example of using MK for proving is:
Prove(Lemma(“Plus is commutative”), Using \rightarrow Theory(“Natural numbers”), By \rightarrow InductionProver), and similarly one can compute (simplify) and solve certain expressions using an exactly specified theory.

3.6 Conclusions

There are many techniques for organising and presenting mathematical knowledge. These techniques are not always accepted or used by the mathematicians and it remains a chal-

lenge to find good presentation vehicles which enable a new era of collaboration between mathematicians and computer scientists. Furthermore, the different systems are each made in a different framework and it is not easy to plug different systems together using their results simultaneously. It is important to bridge different researchers and users (e.g., mathematicians, computer scientists, logicians) and different systems.

4 Algebraic Specification methods and tools

4.1 Motivation and historical remarks

The development of algebraic specification methods started in the mid seventies of the last century with the goal to formally specify data structures used in programming and computer science. The term *Abstract Data Type (ADT)* was coined and algebraic specification became a major research topic in the field of mathematical foundations of software development. The results of this phase are mirrored, for example, in a two volume textbook by Ehrig and Mahr⁶. Meanwhile, the range of application has been extended to the precise specification of complete software systems. Algebraic methods were developed to support the whole software development process, from the requirement definitions to the running programs. An IFIP State-of-the-Art report⁷ shows these developments up to 1998.

The notion of *abstract data type* is closely related to what mathematicians call a mathematical model or structure or domain etc. We therefore believe that algebraic specification methods can serve as a basis for specifying all kinds of mathematical domains in a way suitable for mathematical knowledge management. Moreover, many aspects of representing mathematical knowledge formally, like structuring and parametrization of domains, logic-independent representations, etc. have been studied extensively and these results of software science are ready to use by the MKM community.

4.2 CASL

Many (more than 20, at least) algebraic specification languages have been developed during the last quarter of the last century, most of them coming with a set of tools for using them. This proliferation was a substantial obstacle to the dissemination and use of these techniques. In 1995 the *Common Framework Initiative for algebraic specification and development*, CoFI, was started to unify the various approaches. The specification language developed by CoFI is called CASL⁸: The Common Algebraic Specification Language. CASL has a clear formal semantics based on logic and category theory and essentially is a *family of languages*. It is both, restrictable to sublanguages (for example to achieve more efficiency of generated programs), and extendable to higher-order, state-based, concurrent, etc. These features make CASL the pragmatic choice for using algebraic specification techniques for MKM.

4.3 The CASL Tools Set (CATS)

The CASL Tools Set (CATS) was developed mainly at the University of Bremen in the frame of CoFI initiative. For downloading the system one should visit the Cats homepage :

⁶H. Ehrig, B. Mahr: *Fundamentals of Algebraic Specification*, EATCS Monographs on Theoretical Computer Science, Vol. 6 and 21, Springer-Verlag Berlin, 1985 and 1990

⁷E. Astesiano, H.J. Kreowski, B. Krieg-Brückner (Eds.). *Algebraic Foundations of System Specification*, Springer-Verlag Berlin, 1999

⁸Common Algebraic Specification Language, see: Mosses,P.D., CoFI: The Common Framework Initiative for Algebraic Specification and Development. TAPSOFT '97: Theory and Practice of Software Development (ed. Dauchet,M. & Bidoit,M.) Lecture Notes in Computer Science **1214**, Springer Verlag, 1997, pp. 115–140.

[//www.informatik.de/cofi/CASL/CATS/](http://www.informatik.de/cofi/CASL/CATS/). There on can find a link to contact the authors: The Bremen CASL Team (Till Mosawowski, Markus Roggenbach, Lutz Schröder, Pascal Schmidt).

CATS is written in Standard ML. One can download a standalone version (i.e. Standard ML does not have to be installed) for free. It is available for the Linux and Sun Solaris operating systems, not for Microsoft Windows. There is also a Web-based interface where one can syntax-check CASL specifications remotely. CATS is still under development and not yet an industrial-strength tool.

CATS includes a parser for the full CASL syntax (including libraries), static analysis of full CASL, and produces several types of encodings, helping to interface CASL with existing theorem proving and rewriting tools. Formatting of specifications in LaTeX and OMDoc is also possible. These encodings make it a suitable tool to be used in an environment for formal MKM, i.e. to be used with systems like Isabell/HOL, PVS, Inka, which are also described in this document.

4.4 Other Tools and Methods

In Annex I of the MKMNet proposal (page 19) several other methods and tools are listed to be assessed in this survey. After some research and rethinking of the reasons why they were listed here we decided to deliberately not to include them here, for the reasons given below.

For formal software specification and development methods like Z, B, VDM, etc. the future relevance does not seem to be clear. Even if mathematical methods for software development should play an important role in the application of mathematics, it is all but settled which types of methods this will be. At a recent conference on “Formal Methods for Object and Component Systems”⁹ we got the feeling that people are still seeking agreement on the formal definition of basic notions used in the development of software for today’s mobile, distributed, ubiquitous, wireless, etc. applications. There also seems to be a shift in importance from classical prover-based methods to model checking methods.

Using algebraic specification methods for the development of whole industry-size software systems should be seen with the same reservations about future relevance. At the conference mentioned above algebraic specifications were mentioned only for defining the basic data structures used in large systems. This is not to say that tools like MAYA¹⁰ are not relevant for MKM. Just on the contrary, if algebraic specifications are taken as one basis for MKM, it could be the tool of choice for organizing the complex database of algebraic specifications of all kinds of mathematical domains.

4.5 Assessment and Conclusions

We believe that algebraic specification methods can serve as a basis for specifying all kinds of mathematical domains in a way suitable for mathematical knowledge management. Moreover, many aspects of representing mathematical knowledge formally, like structuring and parametrization of domains, logic-independent representations, etc. have been studied

⁹<http://fmco.liacs.nl/fmco02.html>

¹⁰Described in the next section

extensively and these results of software science are ready to use by the MKM community. It would be extremely unwise to ignore these results developed during the last, say, 25 years.

Specifications of mathematical domains should be written in CASL - The Common Algebraic Specification language. We propose that in a case study a specification of considerable size should be developed. This case study should also be used to find out which extensions or restrictions of the CASL language are appropriate for the special purpose of MKM, perhaps in contrast to the use in formal software development which was to primary goal of CASL. The case study on Hilbert spaces proposed in Annex I of the MKMNet proposal (page 20, responsible: Risc-Linz and Bialystok) seems to be quite appropriate to reach this goal.

The CASL Tools Set (CATS) which is still under development and available without charge should be used to carry out the experiments in the case study proposed above. These experiments should also assess the practicability of using CASL in connection with other tools used in other case studies of WP4. They should also help to find out which enhancements of CATS are needed to make it a tool that we can propose to be used in a FP6 project on MKM.

How big the future relevance of algebraic specification methods will be for the formal development of large software systems, and thus – indirectly – for MKM, is not clear to us at the moment. We propose that this issue should be further discussed and investigated throughout the development of the current project.

5 Automated Reasoning Systems

5.1 First-Order Theorem Provers

Computational logic has a similar long history. Many advanced ATP (HOL, Isabelle, LP, PVS, NQ-THM, OTTER, NuPrl etc) have been developed, addressing both foundational logical issues and practical ones such as search and proof management. They have been particularly successful in specialised domains such as hardware verification, but despite initial optimism have failed to become widely used by non-specialists. The proposers have shown ATP based on rewriting and induction, such as LP, to be particularly valuable to algebraists, and have incorporated these ideas in standard group theory software. Recent significant advances include constrained rewriting, induction, and the advent of more flexible and powerful ATP such as LP and PVS.

Vampire (<http://www.cs.man.ac.uk/~riazanoa/Vampire>) is a resolution based system for completely automatic theorem proving in first-order classical logic with equality. It solves problems in the TPTP syntax in both CNF and full first-order logic syntax. Vampire supports the following inference rules: ordered binary resolution with negative selection, superposition, and a special form of splitting. Vampire exploits a number of redundancy control and simplification techniques, such as forward and backward subsumption, forward and backward demodulation, and forward subsumption resolution. The efficiency of Vampire is due to a combination of compilation and indexing techniques used to implement all costly operations. Code trees are used for forward subsumption and search for generalisations, combination of path indexing and database style joins for backward subsumption and demodulation, and a number of specialised versions of discrimination trees for resolution and superposition.

PROTEIN (<http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN>) is an automated theorem prover for first-order clause logic. It follows Loveland's Model Elimination procedure and is implemented according to Stickel's PTP-Technique, i.e. the input specification is compiled into a Prolog program. Besides "Model Elimination", PROTEIN features various versions of "Restart Model Elimination". Furthermore, PROTEIN has an interface for including theory knowledge into the proving process. Another feature is the computation of the negation of a literal according to WGCWA or GCWA. For theory reasoning within PROTEIN a so-called LC tool is provided that transforms a Horn theory into an efficient background reasoner according to the method of linearizing completion. This tool is optional. In addition to the LC a front-end for automatic detection of theories, called SCAN-IT, is available. It searches for Horn theories, which are automatically linear completed by the LC. PROTEIN was written in ECRC's Prolog-dialect ECLiPSe.

3TAB (<http://i12www.ira.uka.de/~beckert/3tap.html>) is a many-valued tableau-based theorem prover developed at the University of Karlsruhe. It can be instantiated for arbitrary finitely-valued first-order logics, and it can handle equality (two-valued), sorts, and non-clausal input. For equality handling in the two-valued version, uses a completion-based method to solve mixed universal and rigid E-unification problems, that are extracted from tableau branches. (The implementation of this method for equality handling can be used

stand-alone.) In the two-valued version a restricted version of the dissolution inference rule of Murray & Rosenthal is implemented and may optionally be switched on. 3TAB is implemented in Prolog and C.

Waldmeister (<http://www.mpi-sb.mpg.de/hillen/waldmeister>) is a theorem prover for first order unit equational logic. It is based on unfailing Knuth-Bendix completion employed as proof procedure. When searching for a proof, Waldmeister saturates the given axiomatization until the goals can be shown by narrowing or rewriting. The saturation is performed in a cycle working on a set of waiting facts (critical pairs) and a set of selected facts (rules). Inside the completion loop, the following steps are performed: 1. Select an equation from the set of critical pairs. 2. Simplify this equation to a normal form. Discard if trivial, otherwise orient if possible. 3. Modify the set of rules according to the equation. 4. Generate all new critical pairs. 5. Add the equation to the set of rules. The selection is controlled by a top-level heuristic maintaining a priority queue on the critical pairs. This top-level heuristic is one of the two most important control parameters. The other one is the reduction ordering to orient rules with. There is some evidence that the latter is of even stronger influence.

Bliksemis (<http://www.mpi-sb.mpg.de/nivelle/bliksem>) a theorem prover that uses resolution with paramodulation. It is written in portable C. The purpose of Bliksem was to develop a theorem prover that is theoretically up to date, but first order logic and in modal logics, by translating formulae in to clauses. It supports different transformations to clausal normal form. Bliksemis intended both for stand alone use, and for use in combination with an interactive theorem prover. In order to obtain this goal Bliksem is able to output full detailed proofs, which can be verified by another program, or translated into another calculus. The Bliksem project is terminated.

Gandalf (<http://www.ttu.ee/it/gandalf>) is used as a name for a family of theorem provers, currently including versions for classical 1-st order logic, intuitionistic 1-st order logic, propositional linear logic and a subset of Martin L of's type theory. These provers share large parts of their code. Gandalf implements a large number of various search strategies. The usage of these strategies can be either controlled by the human user or by the powerful automatic mode of Gandalf. The automatic mode first selects a set of different strategies which are likely to be useful for a given problem and then tries all these strategies one after another. Gandalf is also well optimised for handling problems where big amounts of *long* clauses are derived. Gandalf is not interactive - it reads in a file containing the problem, outputs information about the progress of proof search and eventually outputs a detailed proof. Gandalf is written in Scheme and compiled to C by the Scheme-to-C compiler Hobbit developed by the author of Gandalf. Gandalf has been ported to a number of UNIX, MS-DOS and Windows platforms.

Otter (<http://www-unix.mcs.anl.gov/AR/>) is an automated resolution prover for first-order logic. The current automated deduction system Otter is designed to prove theorems stated in first-order logic with equality. Otter's inference rules are based on resolution and paramodulation, and it includes facilities for term rewriting, term orderings, Knuth-Bendix

completion, weighting, and strategies for directing and restricting searches for proofs. Otter can also be used as a symbolic calculator and has an embedded equational programming system. Otter is a fourth-generation Argonne National Laboratory deduction system whose ancestors (dating from the early 1960s) include the TP series, NIUTP, AURA, and ITP. Currently, the main application of Otter is research in abstract algebra and formal logic. Otter and its predecessors have been used to answer many open questions in the areas of finite semigroups, ternary Boolean algebra, logic calculi, combinatory logic, group theory, lattice theory, and algebraic geometry. Otter can be run on UNIX, MS-DOS and Windows platforms.

MACE is a program that searches for finite models of first-order and equational statements. For example, if you give it the axioms for a group and state that there are two noncommuting elements, it will produce a noncommutative group. MACE serves as a complementary companion to Otter, which searches for refutations of the same class of statement. In particular, if you have a first-order conjecture, Otter will search for a proof, and MACE will search for a counterexample from the same input file. MACE's engine is a Davis-Putnam-Loveland-Logeman (DPLL) propositional decision procedure. It is available in the Otter-3.2/MACE-2.0 package as a standalone program called ANLDP. Some details of the DPLL implementation can be found in a 1994 report *A Davis-Putnam Program and Its Application to Finite First-Order Model Search: Quasigroup Existence Problems*.

EQP (<http://www-unix.mcs.anl.gov/AR>) is an automated theorem proving program for first-order equational logic. Its strengths are good implementations of associative-commutative unification and matching, a variety of strategies for equational reasoning, and fast search. It seems to perform well on many problems about lattice-like structures. Most recently, it was used to answer the Robbins algebra problem, which had been open for more than half a century. EQP is not as stable and polished as our main production theorem prover Otter. But it has obtained several interesting results, and we have decided to make it available (including the source code) to everyone, with no restrictions (and of course no warranty). EQP's documentation is not good, but if you already know Otter, you might not have great difficulty in learning to use EQP.

5.2 Inductive Theorem Provers

ACL2 (<http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html>) is an automatic/interactive prover for a large subset of applicative Common Lisp. The ACL2 logic is a first-order logic of total recursive functions providing mathematical induction on the ordinals up to ϵ_0 and two extension principles: one for recursive definition and one for constrained introduction of new function symbols. As a specification language, ACL2 supports modeling of systems of various kinds. An ACL2 function can equally be used to express purely formal relationships among mathematical entities, to describe algorithms, or to capture the intended behavior of digital systems. ACL2 is an automated theorem prover or proof checker. This means that a competent user can utilize the ACL2 system to discover proofs of theorems stated in the ACL2 logic or to check previously discovered proofs. The basic deductive steps in an ACL2-checked proof are often quite large, due to the sophisticated

combination of decision procedures, conditional rewriting, mathematical and structural induction, propositional simplification, and complex heuristics to orchestrate the interactions of these capabilities. Unlike some automated proof systems, ACL2 does not produce a formal proof. The ultimate result of an ACL2 proof session is a collection of “events,” possibly grouped into “books,” that can be replayed in ACL2. Therefore, a proof can be independently validated by any ACL2 user. ACL2 may be used in purely automated mode in the shallow sense that conjectures are submitted to the prover and the user does not interact with the proof attempt (except possibly to stop it) until the proof succeeds or fails. However, any non-trivial proof attempt is actually interactive, since successful proof “events” influence the subsequent behavior of the prover. Also, ACL2 supports annotating a theorem with “hints” designed to guide the proof attempt. By supplying appropriate hints, the user can suggest proof strategies that the prover would not discover automatically. There is a “proof-tree” facility (see proof-tree) that allows the user to monitor the progress and structure of a proof attempt in real-time. Exploring failed proof attempts is actually where heavy-duty ACL2 users spend most of their time. ACL2 can also be used in a more explicitly interactive mode. The “proof-checker” subsystem of ACL2 allows exploration of a proof on a fairly low level including expanding calls of selected function symbols, invoking specific rewrite rules, and selectively navigating around the proof. This facility can be used to gain sufficient insight into the proof to construct an automatic version, or to generate a detailed interactive-style proof that can be replayed in batch mode.

CLAM (<http://dream.dai.ed.ac.uk/software/systems/lambda-clam>) is an automated tactical theorem prover with proof planning, inductive and meta-level reasoning. LambdaCLAM is a tool for automated theorem proving in higher order domains. In particular LambdaCLAM specialises in proof using induction based on the rippling heuristic. LambdaCLAM is a higher-order version of CLAM. Both CLAM and LambdaCLAM use proof planning to guide the search for a proof

Proof planning was first suggested by Bundy. A proof plan is a proof of a theorem at some level of abstraction presented as a tree. Each node in this tree is justified by a tactic. The exact nature of these tactics is unspecified, they may be sequences of inference rules, programs for generating sequences of inferences or a further proof plan at some lower level of abstraction. In principle while the generation of the proof tree may have involved heuristics and (possibly) unsound inference steps, it can be justified by executing the tactics attached to the nodes.

A proof plan is generated using AI-style planning techniques (in LambdaCLAM this is basically depth-first search though there is no theoretical reason why this should be the case). The planning operators used by a proof planner are called proof methods these are defined by their pre- and post-conditions which are used by the planner to form the proof plan. Proof methods have been called partial tactic specifications - in theory the method’s pre- and post-conditions describe (partially) the proof state before and after the application of their associated tactic in some meta-language. It is possible to be satisfied with a proof plan as a proof if you are prepared to accept the soundness of the methods - alternatively the tactics can be executed to produce a proof in some object logic as a sequence of axioms and inference rule applications (methods are only infrequently of a sufficiently low level to be considered as inference rule applications).

LambdaCLAM came about when it was decided that we would increasingly need to construct plans over a higher-order domain. This is particularly necessary for program synthesis where the synthesised program starts as an uninstantiated meta-variable, which is gradually instantiated by the proof steps, until a complete proof has been constructed, at which time the meta-variable is a complete program, and the proof is a verification proof that it satisfies its specification.

INKA/MAYA Originally developed as an automatic inductive theorem prover based on resolution and paramodulation, the INKA system was redesigned in INKA 4.0 in the early '90s to meet the requirements arising from its designated use in formal methods. The new version of INKA is a result of this long experience made in formal software development. The major improvements of INKA 5.0 are concerned with the requirements arising when dealing with large applications. The user database is distributed along different deductive units each of which consists of an individual logic (consequence relation) and a set of (local) axioms. In order to allow for the logical implementation of structured specifications as they are provided in languages like CASL, the deductive units may import the deductive reasoning of other units with the help of morphisms. Relationships between different units are also postulated with the help of morphisms between two units which give rise to various proof obligations. INKA also supports the evolutionary aspect of formal software development as it incorporates a management of change. It minimizes the proof obligations arising when changing a deductive unit or defined relationships between some units. As a basis for the implementation of different logics, INKA provides an annotated λ -calculus as an underlying meta-language. Annotations are a generalization of the colour calculus are used to incorporate domain knowledge into the proof search process. INKA provides a uniform hierarchical proof datastructure and a generic tactic definition mechanism to implement appropriate proof search engines.

5.3 Higher-Order and Type Theory Proof Systems

PVS (<http://pvs.csl.sri.com>) provides mechanized support for formal specification and verification. The PVS theorem prover provides a collection of powerful primitive inference procedures that are applied interactively under user guidance within a sequent calculus framework. The primitive inferences include propositional and quantifier rules, induction, rewriting, and decision procedures for linear arithmetic. The implementations of these primitive inferences are optimised for large proofs: for example, propositional simplification uses BDDs, and auto-rewrites are cached for efficiency. User-defined procedures can combine these primitive inferences to yield higher-level proof strategies. Proofs yield scripts that can be edited, attached to additional formulas, and rerun. This allows many similar theorems to be proved efficiently, permits proofs to be adjusted economically to follow changes in requirements or design, and encourages the development of readable proofs.

PVS includes a BDD-based decision procedure for the relational mu-calculus and thereby provides an experimental integration between theorem proving and CTL model checking. PVS is mainly intended for the formalization of requirements and design-level specifications, and for the analysis of intricate and difficult problems. It (and its predecessors) have been chiefly applied to algorithms and architectures for fault-tolerant flight control systems, and to problems in hardware and real-time system design. Several examples are described

in papers listed on the PVS home page. Collaborative projects involving PVS are ongoing with NASA and several aerospace companies; applications include a microprocessor for aircraft flight-control, diagnosis and scheduling algorithms for fault-tolerant architectures, and requirements specification for portions of the Space Shuttle flight-control system. The system is freely available under license from SRI.

Isabelle (<http://www.cl.cam.ac.uk/Research/HVG/Isabelle/>) can be viewed from two main perspectives. On the one hand it may serve as a generic framework for rapid prototyping of deductive systems. On the other hand, major existing logics like Isabelle/HOL provide a theorem proving environment ready to use for sizable applications. Isabelle's Logics The Isabelle distribution includes a large body of object logics and other examples (see the Isabelle theory library). Isabelle/HOL is a version of classical higher-order logic resembling that of the HOL System. Isabelle/HOLCF adds Scott's Logic for Computable Functions (domain theory) to HOL. Isabelle/FOL provides basic classical and intuitionistic first-order logic. It is polymorphic. Isabelle/ZF offers a formulation of Zermelo-Fraenkel set theory on top of FOL. Isabelle/HOL is currently the best developed object logic, including an extensive library of (concrete) mathematics, and various packages for advanced definitional concepts (like (co-)inductive sets and types, well-founded recursion etc.). The distribution also includes some large applications, for example correctness proofs of cryptographic protocols (HOL/Auth) or communication protocols (HOLCF/IOA). Isabelle/ZF provides another starting point for applications, with a slightly less developed library. Its definitional packages are similar to those of Isabelle/HOL. Untyped ZF provides more advanced constructions for sets than simply-typed HOL. There are a few minor object logics that may serve as further examples: CTT is an extensional version of Martin-Löf's Type Theory, Cube is Barendregt's Lambda Cube. There are also some sequent calculus examples under Sequents, including modal and linear logics. Again see the Isabelle theory library for other examples. Logics are not hard-wired into Isabelle, but formulated within Isabelle's meta logic: Isabelle/Pure. There are quite a lot of syntactic and deductive tools available in generic Isabelle. Thus defining new logics or extending existing ones basically one has to declare concrete syntax (via mixfix grammar and syntax macros), declare abstract syntax (as higher-order constants), declare inference rules (as meta-logical propositions), instantiate generic automatic proof tools (simplifier, classical tableau prover etc.), and manually code special proof procedures (via tacticals or hand-written ML). The first three steps above are fully declarative and involve no ML programming at all. Thus one already gets a decent deductive environment based on primitive inferences (by employing the built-in mechanisms of Isabelle/Pure, in particular higher-order unification and resolution). For sizable applications some degree of automated reasoning is essential. Instantiating existing tools like the classical tableau prover involves only minimal ML-based setup. One may also write arbitrary proof procedures or even theory extension packages in ML, without breaching system soundness (Isabelle follows the well-known LCF system approach to achieve a secure system).

HOL - system (<http://www.cl.cam.ac.uk/Research/HVG/HOL/HOL.html>) is an environment for interactive theorem proving in a higher-order logic. Its most outstanding feature is its high degree of programmability through the meta-language ML. The system

has a wide variety of uses from formalizing pure mathematics to verification of industrial hardware. Academic and industrial sites world-wide are using HOL. The name ‘HOL’ is pronounced either as a word to rhyme with ‘doll’ or letter by letter. The system is available without charge. Current work on HOL concentrates on linking the theorem prover to other programs. These include industrial CAD and CASE tools as well as a variety of automatic proof procedures. A prerequisite for combining formal verification with modern hardware design methods is an accurate semantic understanding of hardware description languages (HDLs). We are collaborating with Synopsys, the industry leader in hardware synthesis, to develop a formal semantics of Verilog HDL. Our long-term aim is to provide a flexible core proof engine that can be embedded in other CAD/CASE systems to provide a programmable platform for formal verification.

IMPS (<http://imps.mcmaster.ca>) is an interactive proof system based on a nonconstructive version of simple type theory with partial functions and subtypes, with a theory interpretation mechanism.

IMPS is an Interactive Mathematical Proof System intended to provide organizational and computational support for the traditional techniques of mathematical reasoning. In particular, the logic of IMPS allows functions to be partial and terms to be undefined. The system consists of a database of mathematics (represented as a network of axiomatic theories linked by theory interpretations) and a collection of tools for exploring, applying, extending, and communicating the mathematics in the database. To get a feel for the kinds of mathematics that it is possible to do in IMPS, see the description of the IMPS Theory Library. One of the chief tools of IMPS is a facility for developing formal proofs. In contrast to the formal proofs described in logic textbooks, IMPS proofs are a blend of computation and high-level inference. Consequently, they resemble intelligible informal proofs, but unlike informal proofs, all details of an IMPS proof are machine checked.

VSE (Verification Support Environment) (<http://www.dfki.de/vse>) is a tool to formally specify and verify complex systems. It provides means to structure specifications and supports the development process from the specification of a system to the automatic generation of code. Formal developments following the VSE method are stored and maintained in an administration system that guides the user and maintains a consistent state of the development. An integrated deduction system provides proof support for the deduction problems arising during the development process. Similar to the B-tool it supports a formal development, starting with a formal specification and ending with an automated program code generation from executable specifications. It provides an adequate management of software developments, an elaborate proof construction mechanism, and a sophisticated user interface. Unlike the B-tool or the VSE-I system which supports only the development of sequential programs, the VSE-II system is extended with respect to comprehensive methods in order to deal with distributed and concurrent systems using a TLA-like specification language. The correctness management supports an evolutionary software development which allows changes without the need to prove everything from scratch.

TPS (the “Theorem Proving System”) is an automated theorem-prover for first-order logic and type theory. TPS runs in Common Lisp. TPS has been used extensively under

Unix and Linux systems, and to some extent under Windows.

TPS can be used to prove theorems of first- and higher-order logic interactively, automatically, or in a mixture of these modes, though in automatic mode it is quite primitive in certain respects, such as in dealing with equality. It has facilities for searching for expansion proofs, translating these into natural deduction proofs, constructing natural deduction proofs, translating natural deduction proofs which do not contain cuts into expansion proofs, and solving unification problems in higher-order logic. It has a formula editor which facilitates constructing new formulas from others already known to TPS, and a library facility for saving formulas, definitions, and modes (groups of flag settings).

The interactive facilities of TPS for constructing natural deduction proofs have been used under the name ETPS in logic courses at Carnegie Mellon for a number of years. Students generally learn to use ETPS fairly quickly just by reading the manual (which contains some complete examples) and playing with the system. The student using ETPS issues commands to apply rules of inference in specified ways, and the computer handles the details of writing the appropriate lines of the proof and checking that the rules can be used in this way. The program thus allows students to concentrate on the essential logical problems underlying the proofs, and it gives them immediate feedback for both correct and incorrect actions. ETPS permits students to work forwards, backwards, or in a combination of these modes, and provides facilities for rearranging proofs, deleting parts of proofs, displaying only those parts of proofs under active consideration, saving incomplete proofs, and printing proofs on paper. The convenient formula editor permits the student to extract needed formulas which occur anywhere in the proof, and build new formulas from them. The rules of inference and predefined problems in ETPS are mostly taken from the textbook: Peter B. Andrews, *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, Second Edition, Kluwer Academic Publishers, 2002. Descriptions of the rules of inference are available online. When the teacher permits it, the student can obtain hints from ETPS concerning what to try in various situations. Students can send remarks or questions to the teacher without leaving the program. A record of completed exercises is maintained by ETPS, and can be processed by the GRADER program which is also part of TPS, and which can be used to maintain and process numerical or letter grades for any course.

OMEGA (<http://www.ags.uni-sb.de>) is an interactive proof development system and a mathematical assistant tool that supports proof development in mathematical domains at a user-friendly level of abstraction. It is a modular system with a central data structure and several complementary subsystems. OMEGA has many characteristics in common with systems like NURPL, COQ, HOL, and PVS. However, it differs from these systems with respect to its focus on *proof planning* and in that respect it is similar to the systems at Edinburgh CLAM system. Special features of OMEGA include (1) facilities to access a considerable number of different reasoning systems and to integrate their results into a single proof structure, (2) support for interactive proof development through some non-standard inspection facilities and guidance in the search for a proof, and (3) methods to develop proofs at a knowledge-based level. The OMEGA system combines interactive and automated proof construction in *mathematical domains*. OMEGA's inference mechanism is an interactive theorem prover based on a higher-order natural deduction (ND) variant of

a sorted version of Church's simply typed λ -calculus. The user can interactively construct proofs directly at the calculus level or at the more abstract level of *tactics* and *methods*. Proof construction can be supported by already proven assertions and lemmata and by calls to external systems to simplify or solve subproblems. At the core of OMEGA is the *proof plan data structure* PDS in which proofs and *proof plans* are represented at various levels of granularity and abstraction. The proof plans are classified with respect to a taxonomy of mathematical theories, which are currently being replaced by the mathematical data base MBASE. The user of OMEGA, or the proof planner MULTI, or the suggestion mechanism OANTS modify the PDS during proof development. They can invoke external reasoning systems whose results are included in the PDS after appropriate transformation. After expansion of these high level proofs to the underlying ND calculus, the PDS can be checked by OMEGA's proof checker. User interaction is supported by a graphical user interface LOUI and the proof explainer PREX.

Twelf is a meta-logical framework for the specification, implementation, and meta-theory of deductive systems from the theory of programming languages and logics. It relies on the LF type theory and the judgments-as-types methodology for specification, a constraint logic programming interpreter for implementation, and the meta-logic M2 for reasoning about object languages encoded in LF. It is an extension and complete reimplementaion of the Elf system. Twelf is written in Standard ML and runs under SML of New Jersey and ML Works on Unix and Window platforms. The Twelf implementation comprises the LF logical framework, including type reconstruction; the Elf constraint logic programming language, an inductive meta-theorem prover for LF, and an Emacs interface. Twelf provides a uniform meta-language for specifying, implementing, and proving properties of programming languages and logics. Example suites include Cartesian Closed Categories and lambda-calculus, the Church-Rosser theorem for the untyped lambda-calculus, Mini-ML including type preservation and compilation, cut elimination, theory of logic programming, and Hilbert's deduction theorem.

Coq (<http://pauillac.inria.fr/coq/coq-eng.html>) is a Proof Assistant for a Logical Framework known as the Calculus of Inductive Constructions. It allows the interactive construction of formal proofs, and also the manipulation of functional programs consistently with their specifications. It runs as a computer program on many architectures, and mainly on Unix machines. It is available with a variety of user interfaces. The system Coq is designed to write formal specifications, programs and to verify that programs are correct with respect to their specification. It provides a specification language named Gallina. Terms of Gallina can represent programs as well as properties of these programs and proofs of these properties. Using the so-called Curry-Howard isomorphism, programs, properties and proofs are formalized the same language called Calculus of Inductive Constructions, that is a λ -calculus with a rich type system. All logical judgments in Coq are typing judgments. The very heart of the Coq system is the type-checking algorithm that checks the correctness of proofs, in other words that checks that a program complies to its specification. Coq also provides an interactive proof assistant to build proofs using specific programs called tactics. Coq has an interactive mode in which commands are interpreted as the user types them in from the keyboard and a compiled mode where commands are processed from a

file. Other modes of interaction with Coq are possible, through an emacs shell window, or through a customized interface with the Centaur environment (CTCoq). The compiled mode acts as a proof checker taking a file containing a whole development in order to ensure its correctness. Moreover, Coq's compiler provides an output file containing a compact representation of its input. The compiled mode is run with the `coqc` command from the operating system.

NUPRL (<http://www.cs.cornell.edu/Info/Projects/NuPrl/html/NuprlSystem.html>) is a proof system for an intuitionistic type theory based on Martin Lof type theory, with proof by refinement and extraction of programs from proofs. Oyster is an Edinburgh implementation of the Nuprl System. The Nuprl proof development system is a framework for the development of formalized mathematical knowledge as well as for the synthesis, verification, and optimization of software. It is based on a significant extension of Martin-Löf's intuitionistic Type Theory, which includes formalizations of the fundamental concepts of mathematics, data types, and programming. The system itself supports interactive and tactic-based reasoning, decision procedures, evaluation of programs, language extensions through user-defined concepts and an extendable library of verified knowledge from various domains. Since its first release in 1984, the system has been undergone several significant modifications to meet the growing demands for formal knowledge and tools in programming and mathematics. The Nuprl 5 system features an open, distributed architecture that integrates all its key subsystems as independent components and uses a flexible knowledge base as its central component.

Theorema¹¹ ([://www.theorema.org](http://www.theorema.org)) is somewhat difficult to include in this classification, since it combines techniques from first order logic as well as (elementary inferences) from higher-order logic, with induction proving and other proving techniques, and also has a special human-oriented interface.

Theorema is a prototype system (still under development) for computer aided mathematics, thus it also has other components, but from the proving point of view it exhibits a collection of various provers, with more being promised in the future. This collection includes: propositional prover, first-order predicate prover (with some elementary inferences for higher-order logic), PCS ("Proving-Computing-Solving") prover for the limit domain, specific provers for the domains of sets, tuples (simple induction), integers (simple induction), as well as a number of provers which use specific algebraic techniques as "black box" sub-algorithms (Groebner Bases, Combinatorial telescoping).

All provers of Theorema are based on inference rules which are similar to the ones used by human provers ("natural style"). Correspondingly, the generated proofs are displayed with natural language explanations.

5.4 Human Computer Interfaces to Theorem Provers

The practical utility of theorem proving depends on good interface design, the study of which should be firmly grounded in the best available methods of research into Human-Computer Interaction. There are several projects and individual researchers working on

¹¹Paragraph contributed by Tudor Jebelean.

interfaces or principles for interface design. Most of the tool support developed so far is specific to individual theorem provers. Typically these approaches follow the direct manipulation paradigm (Shneiderman,87) visualizing the proof object and allowing to manipulate the proof object by simple interactions like mouse clicks or simple keystrokes. In the following some approaches are exemplarily listed. Many of these features have migrated also to other tools (listed above) in the meanwhile.

JAPE (Just Another Proving Environment) (<http://www.dcs.qmul.ac.uk/~richard/jape>) is intended as a flexible interface to interactive theorem proving environments. Basically, JAPE is able to present logical proofs in a number of forms: tableau, list, tree. Users can build a proof in any of these formats using direct manipulation user interfaces. The results is a fully graphical user interface to a formal logical proof. JAPE is not committed to any particular logic - the symbols and inference rules of a particular logic can be defined. Furthermore, for a given logic it is possible to define ways of presenting the inference rules to the user. However, JAPE is restricted to the visualization of calculus rules and does not support the presentation of tactical theorem proving or proof planning. JAPE has been used mostly in undergraduate teaching of formal logic to computer science students. However, this had provided a lot of insight into user needs and how users interact with logical proofs. The tool is freely available.

NuPrI Web Publisher (<http://www.cs.cornell.edu/home/pavel/WebPublisher>) is designed to make NuPrI theories accessible through the web. Essentially it is a very simple tool that produces a page for each theory. On the page is the name and statement of each theorem in the theory. Clicking on the appropriate links reveals the term structure of the theorem which is hyperlinked to appropriate definitions elsewhere in the theory.

XBarnacle (<http://www.dream.dai.ed.ac.uk/software/systems/lambda-xbarnacle>) is a version of CLaM automated proof planner incorporating a graphical user interface that allows users to interact with CLaM during a proof. XBarnacle is designed to allow users to step in and use their domain knowledge to guide CLaM in the search for a proof. This might be appropriate if they conclude that CLaM is pursuing an unproductive search strategy or CLaM performs a proof step the user knows is unproductive. XBarnacle features an implementation of interactive proof critics which provide functionality to CLaM to allow the patching of failed proof steps allowing then to succeed. Part of the functionality of the interactive proof critics is an explanation facility which describes why a method failed in terms of its preconditions, why a critic was applicable, in terms of failure of the associated methods preconditions, and what the critic will do.

Pcoq provides a working environment for the Coq theorem prover. It has been developed following a general approach for building user-interfaces for theorem provers. It has the following characteristics: Pcoq uses multiple fonts and colors to display formulas and commands. The graphical interface and Coq run as two separate processes. Users may choose which machine to run Coq on in their network of workstations. The environment provides ways to edit structurally formulas and commands, new notations can easily be added, The environment uses the structure of logical formulas to help systematically the

user direct the proof by simple clicks of the mouse. Pcoq is developped and maintained in the Lemme team at INRIA Sophia-Antipolis.

6 Computer Algebra Systems

6.1 Introduction

The purpose of this chapter is to offer a brief overview of the currently available Computer Algebra systems and to make a preliminary assessment of their usability from the point of view of formal Mathematical Knowledge Management. This is only a preliminary assessment, based on the limited experience of the authors in using CA systems for formal MKM and on speculative projections on possible requirements of an MKM system. A more accurate assessment will be provided at the end of the project, based on the experience of the project partners during the development of several concrete test cases (task 4.2 of the workprogramme).

The following *requirements* of Mathematical Knowledge Management in a formal way are considered:

- The possibility to perform computations using the mathematical knowledge, since formal MKM has to allow full manipulation of various formulae.
- The naturalness of the user interface, since the users may be non-specialists in mathematics and, in particular, in Computer Algebra.
- The access on the Internet, since an MKM system has to address a large community of users.

Correspondingly, we identify the following *criteria* for assessing the usability of the various systems:

- Power: is it powerful enough for computations in a large variety of mathematical domains? We include in this survey only those systems, which, in our opinion, offer sufficient power in order to be considered as possible candidates for supporting a formal MKM system.
- Purpose: general or specialized. General purpose systems have the advantage that they support a large variety of computations, therefore they are more appropriate for MKM. Specialized systems provide more power and accuracy for certain mathematical topics, hence they may be used as special computing engines. Since this survey is done at a general level (not having a particular type of mathematical knowledge in mind), we do not discuss specialized systems here, but refer to the general information sources mentioned below, which contain descriptions of several dozens of specialized systems.
- Availability: is it a commercial system or a free software. A commercial system, especially if it has a large user base, has the advantage of stability in time. Free software has the advantage of allowing access to the source code.
- Flexibility: can it be used as a computing engine from external programs, or only stand alone? In the second case the system is usable only if it allows to build an efficient MKM system on top of it – this may require the capability to call external programs from within the CA system.

- Internet: has the system special facilities for interfacing with the WWW? Is this at all possible (in an efficient way)?
- Graphics: to which extent does the system facilitate graphical presentation of mathematical objects?
- 2D formulae: is it possible/easy to use natural 2D input and/or output of mathematical formulae? To which extent can the user influence the style of the 2D expressions?

The importance of these criteria for satisfying the requirements of an MKM system depends of course on the particular usage which the CA software has in the whole MKM system. For instance, the architecture of the MKM system may be organized in such a way that the CA software is used only as a computing engine, and the user interface and/or the Web interface is performed by other tools. On the other way, the implementation and/or prototyping of a formal MKM system may be significantly facilitated by a CA software which offers good capabilities for graphics, 2D formulae, etc.

It is below the scope of this chapter to provide an exhaustive survey of all available Computer Algebra systems. Additional details on Computer Algebra systems are provided in the literature and on the Internet, and we give some references below. The most recent source of information on Computer Algebra Systems is the “Computer Algebra Handbook” by J. Grabmeier, E. Kaltofen, and V. Weispfenning (Springer 2002), which is also probably the most authoritative source on Computer Algebra in general. Another good survey is: “Computer Algebra Systems” by M. J. Wester (Wiley 1999). The most important online sources on information are SymbolicNET <http://www.symbolicnet.org> and CAIN (Computer Algebra Information Network) [://www.cain.nl](http://www.cain.nl). The most important professional organization of specialists in Computer Algebra is SIGSAM/ACM (the Special Interest Group in Symbolic and Algebraic Manipulation of the Association for Computing Machinery) [://www.acm.org/sigsam](http://www.acm.org/sigsam).

6.2 Mathematica

Mathematica is a commercial general-purpose computer algebra system and has the largest user-base (approx. 3 million users).

The system contains all the important algorithms used in computer algebra, as well as a variety of numerical algorithms.

Mathematica interfaces through an interpreted programming language based on conditional rewriting using pattern matching. This allows direct implementation of functional programs described by conditional equalities. An unique feature is the usage of “sequence variables”: pattern variables which can be replaced by an arbitrary number of terms.

The computing engine can be used from other programs through a C/C++ programming interface “mathLink” which is provided with the system. Similarly, programs written in Mathematica can use external C programs. A similar toolkit is also provided for linking with Java programs.

An unique feature is the separation of the system into FrontEnd and Kernel, which can be used also separately: the FrontEnd for input/output of mathematical formulae, and the Kernel as a computing engine.

The native notebooks of Mathematica can be automatically saved in HTML format, but some of the features of the presentation are lost, most notably the scaling of formulae (which are translated into images) and the bracketing/manipulation of cells (closed cells are translated into links). Mathematica supports the translation of formulae into MathML, notably in the frame of *webMathematica*: a novel additional package (has to be purchased separately) which allows the interfacing of web pages with the system (see [://www.wolfram.com/products/webmathematica](http://www.wolfram.com/products/webmathematica))

Full graphical capabilities are provided for the representation of mathematical objects, including 2D plots, 3D plots, and animated figures. The graphics can be automatically exported in all popular formats.

Input and output of formulae is possible in natural 2D representation. An unique feature of Mathematica is the possibility of customization of the 2D appearance of the formulae: the user can describe the box structure which corresponds to a certain function. More recently the system was added the possibility of using user defined graphics in 2D formulae: thus the user can also create new mathematical symbols.

Mathematica is available from *Wolfram Research*, (see www.wolfram.com).

6.3 Maple

Maple is a commercial general-purpose computer algebra system whose popularity is rivaled only by Mathematica.

The system implements a full range of algebraic and numerical algorithms, an unique feature is the possibility of interfacing with MATLAB from NAG, which expands its numerical capabilities.

Maple is an interactive interpretative environment with a procedural high-level language (similar to C and Pascal). It is possible to call C and Fortran programs from Maple. For the reverse interface, one can translate Maple code into C or Fortran code. Most of the algorithms are implemented in Maple language, and the code is open for inspection, which is an unique feature for a commercial system.

The native Maple worksheets can be automatically translated into HTML format, but the formulae are rendered as pictures.

The system provides extensive graphic plotting in 2D, 3D, and also animation. An unique feature is the possibility of altering the point of view for 3D plots in interactive mode (mouse drag).

Usual mathematical formulae are echoed by the system in a natural 2D notation. Input of mathematical text is normally done in the 1D textual notation, but also limited possibility of 2D input is offered, using a special palette of most common mathematical notations. No customization of the 2D formatting is possible.

Maple is available from *Waterloo Maple Inc.* – see www.maplesoft.com.

6.4 Reduce

Reduce is a general-purpose computer algebra system developed and maintained in an academic environment. The system and all its source programs are freely available.

The system implements a large variety of computer algebra and numeric algorithms.

Reduce is implemented in the programming language Lisp (more exactly RLISP). Since the source is available, interaction from/with Reduce is possible from/with any other system, but with the appropriate documentation and programming effort. The system also offers facilities for the generation of partial or complete programs in FORTRAN and C.

There are no specific features for accessing Reduce over the internet.

The system has an interface to GNUPLOT in order to display graphs.

Only textual input/output is available. Usually, the output is echoed in 2D form using text, and no graphics. An interface to the graphical capabilities of system (Windows, X11) is provided for the display of formulae.

It is doubtful whether this system is still actively maintained: the latest version and the latest references we found are from 1999.

6.5 MuPAD

MuPAD is a general purpose computer algebra system, which features a large number of algorithms, both for symbolic and for numerical computations. The system observes the object oriented paradigm, and has a comprehensive collection of types covering a wide range of data structures. New types and data structures can be defined by the user. Particularly interesting from the point of view of Mathematical Knowledge Management is the possibility of defining abstract and generic data types (as e. g. rings, modules), which correspond to algebraic structures from mathematics, using the constructors of *domains* and of *categories*. As in mathematical theories, properties of domains and categories can be axiomatized and can be inherited.

Another unique feature is the possibility of asserting certain properties of objects (using the command *assume*), which are then used for determining the behaviour of certain expressions containing these objects. The type of “formulae” which can be assumed are however limited, they refer to typical properties which are interesting from the algebraic point of view.

The system provides interfaces to C++ and Fortran, in fact the domain structures are compatible with C++ classes. Linking with object code is even possible at run time. Conversely the MuPad programs can be used from C and Java programs, and they are in fact used in various projects, including some which make available the MuPad computing engine to graphical-based interface calculators, textbooks, internet pages, etc.

A large part of the system is open-source code (in MuPad language).

Also as a unique feature, MuPad has a source-level debugger, with a graphical user interface.

Due to its ease of interface with C and Java, MuPad resources can be easily accessible to internet applications. However, the core system itself does not provide specialized functions for this, there is no, for instance, translator of MuPad output into HTML or MathML.

MuPad provides output in 2D and 3D graphics, equaling the performance of other computer algebra systems which have good graphics capabilities.

The output of the system is formatted in the natural 2D notation, defined implicitly for the usual collection of mathematical functions. However, the input of expressions is done in linear form. There are a couple of projects (resulting in MuPad libraries) which allow input into MuPad via ad-hoc graphical editors.

Traditionally developed in an academic environment, MuPad is now distributed and supported via a commercial company (SciFace – www.sciface.com). However, most of the MuPad library are developed in MuPad language itself and their source code is open to the user.

Full information on the system is available at www.mupad.de, including links to various academic and commercial projects which use MuPad.

6.6 Other Systems

Axiom and Aldor An unique feature of Axiom (and later Aldor) is the comprehensive type system which comprises most of the mathematical objects currently in use.

Although Axiom is listed in “Computer Algebra Handbook” (Grabmeier, Kaltofen, Weispfenning - Springer 2002) as being available from NAG (Numerical Algorithms Group, UK), it is in fact not supported by this company anymore (see [://www.nag.com/symbolic/software.asp](http://www.nag.com/symbolic/software.asp)).

The most prominent features of Axiom have been incorporated conceptually in the programming language Aldor (www.aldor.org), and probably some of the algorithms from the Axiom system are incorporated in some of the software tools provided by NAG.

Since NAG is a partner in this project, the possibility of using Aldor and NAG tools for Mathematical Knowledge Management will be closely followed-up, however they are both outside the scope of this survey.

Derive Derive is a commercial general-purpose computer algebra system offering less capabilities than e. g. Mathematica or Maple, but using much less resources. This software was especially designed for use on the portable calculator TI-92, but now is available on PC Windows too, and as such it has a certain success, mostly in high-school education.

However, due to the limitations to a special hardware platform, we appreciate that in the current stage of the MKM project one cannot consider Derive for a prototype implementation.

Additional information is available at [://education.ti.com](http://education.ti.com).

Macsyma Macsyma is a general-purpose system, probably the oldest commercial system available. Until 1998 the system was commercially available from an unique source, which appears to be now out of business (www.macsyma.com). The concepts of the original design have been used in several systems which are now available ([://www.symbolicnet.org/systems/macsyma.html](http://www.symbolicnet.org/systems/macsyma.html)).

Some of these versions are maintained by a single person or a group of enthusiasts (notably DOE MAXIMA [://maxima.sourceforge.net/](http://maxima.sourceforge.net/), which is developed within the GNU project), some others are commercially available on a limited hardware-software platform (e. g. Macsyma 2.4 from ScienTek [://www.scientek.com/macsyma/mxmain.htm](http://www.scientek.com/macsyma/mxmain.htm)) and each of them has presumably a quite low user base.

We appreciate that, for the moment, the usage of one of these tools is not secure in the long run.

6.7 Conclusions

The following table summarizes the most important characteristics of the systems which could be considered for formal MKM purposes:

Name	Availability	Flexibility	Internet	Graphics	2D formulae
Mathematica	commercial	very good	very good	very good	very good
Maple	commercial	very good	average	very good	good
Reduce	free source	good	very weak	very weak	very weak
MuPAD	commercial	very good	good	very good	good

Most suitable appear to be Mathematica and MuPad, followed closely by Maple.

7 More general mechanisms for representing resources

7.1 HELM

Project HELM¹²(A Hypertextual Electronic Library of Mathematics) is aimed at the study and the development of a technological infrastructure for the the creation and maintenance of a virtual, distributed, hypertextual library of *formal* mathematical knowledge. The main leit-motifs of the project are the accessibility of the mathematical repositories in an application independent format, as it can be conveniently provided by XML, and the extensive use of XML-technology for implementing all those generic functionalities such as rendering and searching which, being largely independent from specific logical frameworks, can be conveniently decoupled from the current tools for the automatization of formal reasoning.

In particular, the crucial point is the evolution from the old application-oriented management of information, to a new content-centric architectural design¹³,enabling the exploitation of information in the “Web way”, that is without a central authority, with few basic rules, in a scalable, adaptable, and extensible manner. Establishing a layer of simply accessible and universally understandable data is the key to allow the design of sophisticated search engines and interoperable services, and to enable higher degree of automation and more intelligent applications.

Compared with OpenMath, the main differences between the two projects may be simply summarized by the following table:

	Intended Domain	Main Issue	XML usage
OpenMath	Computer Algebra Systems	Interoperability	Short-term Persistence (interprocess communication)
HELM	Proof Assistants	Rendering, Searching & Retrieving	Long-term persistence (libraries management)

Figure 1: A comparison between OpenMath and HELM

OpenMath is essentially focused on interoperability issues: the aim is to be able to exchange information between different Computer Algebra Systems. The OpenMath markup is mainly meant as a short term serialization format for interprocess communication. The conversion of an OpenMath content object to/from its internal representation in a software application is performed by an interface program called Phrasebook. In turn, the development of Phrasebooks should be directed by Content Dictionaries, which provide human-readable (but not machine-understandable) descriptions of OpenMath objects (they have been explicitly conceived as background documentation for the developers of phrasebooks).

On the contrary, one of the aim of HELM is improving the modularity of the applications, and in particular decoupling all those functionalities which are largely independent

¹²Asperti, A., Guidi, F., Padovani, L., Sacerdoti Coen, C. Schena, I., “Mathematical Knowledge Management in HELM”. Special Issue of the Annals of Artificial Intelligence devoted to MKM - to appear 2003.

¹³Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., “Content-centric Logical Environments”. Short presentation at LICS’2000, June 26-28, 2000, Santa Barbara, California.

form the specific framework used by the application, thus passing from an application-based architectural design to a content-based one, build around the actual content of the information.

We have so far tested HELM approach on the mathematical library of the COQ Proof assistant (comprising about 50000 mathematical statements in various fields of elementary mathematics). The Coq library is accessible on line at [://helm.cs.unibo.it/library.html](http://helm.cs.unibo.it/library.html) Some preliminary experiments have been also done with the mathematical repository of the NuPRL system.

Rendering XML already offers sophisticated Web-publishing technologies, such as Stylesheets, MathML, SVG, Formatting Objects which may be profitably exploited to get the highly demanding stylistic and notational quality required by mathematical documents.

Among the benefits of these technologies, the most important ones for HELM are media independence and notational and stylistic customization. In particular, the combination of client-side customizable rendering languages with the flexibility of user-provided stylesheets can be profitably used to solve, in a *standard, extensible* way the annoying notational problems that traditionally afflict formal mathematics.

In HELM, the transformation of a document from its low-level logical encoding to the intended representation format (currently, we may produce HTML, MathMLpresentation and postscript) is conceived as a complex pipe of stylesheet applications.

In particular, there are two main flows of transformations, according to the two possible outputs, namely theories or individual objects. With theory we mean a collection of objects, possibly intermixed with text and images, structured into sections, chapters and so on. The most complex transformation process obviously concerns objects (and its sublevels: namely propositions, terms and proofs, comprising the required notational support). The interesting problem regarding theories is their dynamic generation; once created, they do not require major transformations, relying on the transformations of the embedded objects.

The transformation of objects is essentially divided in two major phases, passing through an intermediate “content” description. Very roughly, the first phase is responsible for major restructuring at the level of proofs, and the identification of abstract mathematical notions. The second phase, is mainly responsible for the application of notational stylesheets, transforming the intermediate abstract representation into a suitable rendering format.

Searching and retrieval Smart searching and retrieval are not only useful to browse the library, but are also fundamental for the development of Proof Assistants to allow effective re-use of previously developed results; even if this seems a trivial requirement for a Proof Assistant, currently many theorems and definitions are re-stated by the authors in new contributions due to the sheer difficulty to identify the needed notion in the already developed knowledge base.

HELM approach to searching and retrieval is mostly based on a complex set of Metadata, automatically computed and providing an approximate description of the content. Typical examples are forward and backwards pointers enriched with some information about the actual position of the occurrence. The result of the query may be then furtherly refined by other means (e.g. unification).

The Metadata model is expressed in RDF (Resource Description Framework). HELM

provides several RDF Schemas to define both a specific vocabulary for HELM mathematical resources, and a vocabulary for metadata about general resources, say the Dublin Core elements.

All the considered meta-information is associated both to theories/views and to single mathematical entities. Less expensively, though, some metadata is associated to whole collections of mathematical objects (e.g. the author of a bunch of related theorems and definitions). URIs are exploited to associate a RDF file (a metadata model) to the corresponding XML data file or directory (e.g. view, collection of objects, single entity).

After a long and somehow deceiving experimentation with already available languages and tools, we have developed our own query language for RDF information¹⁴

7.2 Mowgli

MOWGLI (Mathematics On the Web: Get it by Logic and Interfaces) is an European Project founded by the European Community in the “Information Society Technologies” (IST) Programme. The partners are the University of Bologna (coordinator), the German Research Center for Artificial Intelligence (DFKI Saarbrücken), the Katholieke Universiteit Nijmegen, the Max Planck Institute for Gravitational Physics (Albert Einstein Institute, TU Berlin), and Trusted Logic (France).

The aim of the project is the study and the development of a technological infrastructure for the the creation and maintenance of a virtual, distributed, hypertextual library of mathematical knowledge based on a *content* description of the information.

Currently, almost all mathematical documents available on the Web are marked up only for presentation, severely crippling the potentialities for automation, interoperability, sophisticated searching mechanisms, intelligent applications, transformation and processing. The goal of MOWGLI is to overcome these limitations, passing from a machine-readable to a machine-understandable representation of the information, and developing the technological infrastructure for its exploitation. MOWGLI builds on previous “standards” for the management and publishing of mathematical documents (MathML¹⁵, OpenMath, OM-Doc¹⁶), integrating them with different XML technology¹⁷ (XSLT¹⁸, RDF¹⁹, SOAP²⁰,...). All these languages cover different and orthogonal aspects of the information and its man-

¹⁴F. Guidi, I. Schena. *A Query Language for a Metadata Framework about Mathematical Resources*. Proceedings of the second International Conference on Mathematical Knowledge Management. Bologna, Italy. February 2003.

¹⁵Mathematical Markup Language (MathML) 2.0 W3C Recommendation, 21 February 2001. [://www.w3.org/TR/MathML2/](http://www.w3.org/TR/MathML2/).

¹⁶Kohlase, M. “OMDoc: Towards an Internet Standard for the Administration, Distribution and Teaching of mathematical Knowledge”. Proceedings of “Artificial Intelligence and Symbolic Computation”, Springer LNAI, 2000.

¹⁷Extensible Markup Language (XML) Specification. Version 1.0. W3C Recommendation, 10 February 1998. [://www.w3.org/TR/REC-xml](http://www.w3.org/TR/REC-xml)

¹⁸XSL Transformations (XSLT). Version 1.0, W3C Recommendation, 16 November 1999. [://www.w3.org/TR/xslt](http://www.w3.org/TR/xslt).

¹⁹Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999.

[://www.w3.org/TR/1999/REC-rdf-syntax-19990222/](http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/)

²⁰SOAP Version 1.2 Part 0: Primer. W3C Working Draft 17 December 2001. [://www.w3.org/TR/2001/WD-soap12-part0-20011217](http://www.w3.org/TR/2001/WD-soap12-part0-20011217).

agement; our aim is not to propose a new standard, but to study and to develop the technological infrastructure required for taking advantage of the potentialities of all of them.

MOWGLI will make an essential use of standard XML technology and aspires to become an example of “best practice” in its use, and a leading project in the new area of the Semantic Web. In particular, we shall deeply explore the potentialities of XML in the following directions:

Publishing XML offers sophisticated publishing technologies (Stylesheets, MathML, SVG, ...) which can be profitably used to solve, in a standard way, the annoying notational problems that traditionally afflict content based and machine-understandable encodings of the information.

Searching and Retrieving Metadata will play a major role in MOWGLI. New W3C languages such as the Resource Description Framework or XML Query are likely to produce major innovative solutions in this field.

Interoperability Disposing of a common, machine understandable layer is a major and essential step in this direction.

Distribution All XML technology is finally aimed to the access of the Web as a single, distributed resource, with no central authority and few, simple rules.

The project deals with problems traditionally belonging to different scientific communities: digital libraries, Web publishing, automation of mathematics and computer aided reasoning. Any serious solution to the complex problem of mathematical knowledge management needs a coordinated effort of all these groups and a synergy of their different expertise. To our knowledge, MOWGLI is the first attempt to build a solid cooperation environment between these communities.

From a machine readable to a machine understandable information The goals of MOWGLI largely overlap with the aims of the so called *Semantic Web*²¹). Associating meaning with content or establishing a layer of machine understandable data will allow automated agents, sophisticated search engines and interoperable services and will enable higher degree of automation and more intelligent applications. The ultimate goal of the Semantic Web is to allow machines to share and exploit knowledge in the Web way, i.e. without central authority, with few basic rules, in a scalable, adaptable, extensible manner. However, the actual development of the Semantic Web and its technologies has been hindered so far by the lack of large scale, distributed repositories of structured, content oriented information. The case of Mathematical knowledge, the most rigorous and condensed form of knowledge, is paradigmatic. The World Wide Web is already now the largest single resource of mathematical knowledge, and its importance will be exponentiated by the emerging display technologies like MathML.

However, almost all mathematical documents available on the Web are marked up only for presentation, making impossible to offer added-value services like

²¹Tim Berner's Lee. *The Semantic Web*. W3C Architecture Note, 1998.

- Preservation of the real informative content in a highly structured and machine understandable format, suitable for transformation, automatic elaboration and processing.
- Cut and paste on the level of computation (take the output from a Web search engine and paste it into a computer algebra system).
- Automatic proof checking of published proofs
- Semantical search for mathematical concepts (rather than keywords).
- Indexing and Classification.

Due to its rich notational, logical and semantical structure, mathematical knowledge is thus a main case study for the development of the new generation of semantic Web systems. The aim of the MOWGLI project is both to help in this process, as well as pave the way towards a really useful virtual, distributed, hyper-textual resource for the working mathematician, scientist or engineer.

MathML, introducing for the first time a content markup layer in parallel with a presentational one, has indubitably been a pioneering project towards the mining of the mathematical treasure available on the web. Still, its limitations are evident as well:

- MathML is merely focused on mathematical *expressions*. However, in order to bring the idea of a Semantic Web of Mathematics to its full potentialities, other layers of mathematical information must be considered as well. In particular, we need a clean, microscopic description of *proofs*, a markup for mathematical *objects* (Theorems, Lemmas, Corollaries, Examples, etc.), a markup for *structured collections* of these objects (Documents, Theories, etc.), possibly *functors* between these collections, and finally a good *metadata* layer.
- MathML is just an (important) piece in a much wider technological puzzle.

Passing from content to a good presentational format requires sophisticated operations; on the other side, these transformations are themselves a basic component of the whole mathematical knowledge (like mathematical fonts). XSLT provides here the right technology, opening the way to the creation of well maintained and documented libraries of mathematical stylesheets²².

Similarly, the creation and maintenance of the library as a distributed repository, and the crucial aspect of managing the information in the “web way”, i.e. with few basic rules and without any central authority, requires a light but powerful communication protocol, overcoming some of the limitations of HTTP (SOAP looks as a promising solution).

Metadata will eventually require a fairly sophisticated model, much beyond what is currently offered by typical metadata models as the Dublin-Core System²³. Here, RDF looks as the right framework for developing the model, providing a general

²²Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., “XML, Stylesheets and the re-mathematization of Formal Content”. Proceedings of “Extreme Markup Languages 2001 Conference”, August 12-17, 2001, Montréal, Canada.

²³The Dublin Core Metadata Initiative. [://purl.org/dc/](http://purl.org/dc/)

architectural model for expressing metadata and a precise syntax for the encoding and interchange of these metadata over the Web.

The fact of encoding also the microscopic, logical level of mathematics opens the possibility to have completely formalised subsystems of the library²⁴, which could be checked automatically by standard tools for the automation of formal reasoning and the mechanisation of mathematics (proof assistants and logical frameworks). At the same time, any of these tools could be used as an authoring system for documents of the library, by simply exporting their internal libraries into XML, and using stylesheets to transform the output into a standard, machine-understandable representation, such as MathML content markup or OpenMath.

An alternative route for the creation of content-based mathematical information from standard digital repositories by means of a suitable LaTeX-based authoring system will be explored by the Albert Einstein Institute (AEI) in Golm (Germany). AEI publishes a solely electronic review journal - Living Reviews in Relativity - on the Web, which provides refereed, regularly updated review articles on all areas of gravitational physics. AEI will develop a LaTeX-based authoring tool interfacing with MOWGLI, and serve as a showcase to demonstrate how content-mark-up in mathematics improves the usability and information depth of electronic science journals.

A minimal technological infrastructure It is clear that the creation and maintenance of large repositories of content-based mathematical knowledge can only be conceived as a cooperative and distributed process, comprising not only the creation of documents, but also libraries of notational rules, metadata and management tools. The crucial point is to build a minimal infrastructure to start up this process, so that more and more tools can be added by interested parties. All these considerations lead to two of main architectural consideration:

- Information must be accessible in the “web way”, that is with few basic rules and no central authority.
- Make extensive use of standard XML technology and tools, even when it would be easier or more efficient just to develop an ad-hoc solution.

In this way, we put no barrier to third party development and, every time a standard technology or tool is improved, we can simply benefit of the new implementation with minimal effort.

The MOWGLI architecture is essentially based on three components, which are distribution sites, standard browsers and plug-outs, and active components, such as XSLT processors, to elaborate the information. Distribution sites are simply HTTP and FTP servers, widespread throughout the world; user browsers are HTTP clients and run on the user host. We do not require any other components to run on a specific host. Active components must provide answers to browsers, requiring an HTTP server interface; they must

²⁴Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., “Formal Mathematics in MathML”. Proceedings of the First International Conference on “MathML and Math on the Web”, October 20-21 2000, University of Illinois at Urbana-Champaign.

also ask data to distribution sites, acting as HTTP clients. Hence, MOWGLI is essentially conceived as an HTTP pipeline.

The module client of the distribution sites is the *getter*, which maps URIs to URLs and hence documents, offering functionalities similar to the APT packet management system ([://www.debian.org](http://www.debian.org)).

The main active component is the XSLT stylesheet manager, whose typical functionality is the application of a list of stylesheets (each one with the respective list of parameters) to a document. However, other components may be added in a completely modular way. This is exactly the content-based architectural design of future web system enabled by XML technology.