efs efs efs spec[**Kohlhase:OMDoc1.6spec**]

Contents

We present a variety of applications and projects that use the OMDoc format or are related to it in a substantive way.

Apart from the projects directly reported here, the OMDoc format is used by the new research field of Mathematical Knowledge Management (MKM; cf. `http://www.mkm-ig.org/`), which combines researchers in mathematics, computer science, and library science. We refer the reader to the proceedings of the annual MKM conference [**MKM01**; **MKM03**; **MKM04**; **MKM05**; **MKM06**].

[1] EdN:1

[1] EdNote: must show the metatdata in project descriptions

Chapter 1

Introduction

The text in the project descriptions has been contributed[1] by the authors marked in the section headings, for questions about the projects or systems, please visit the web-sites given or contact the authors directly. Note that the material discussed in this chapter is under continuous development, and the account here only reflects the state of mid-2006, see `http://www.omdoc.org` for more and current information.

1.1 Overview

The OMDoc format as a whole and the applications mentioned above are supported by a variety of tools for creating, manipulating, and communicating OMDoc documents. We can distinguish four kinds of tools:

Interfaces for Mathematical Software Systems like automated theorem provers. These system are usually add-ins that interpret the internal representation of formalized mathematical objects in their host systems and recursively generate formal OMDoc documents as output and communication streams. Some of these systems also have input filters for OMDoc like the √eriFun described in ?projects@verifun?, but most rely on the OMDoc transformation to their native input syntax described in .

Invasive Editors i.e. are add-ins or modes that "invade" common general-purpose editing systems and specialize them to deal with the OMDoc format. The **C**Point add-in for MS PowerPoint (?projects@cpoint?), the SENTIDO plugin for MOZILLA-based browsers, and the plugin for TEXMACS (?projects@texmacs-omega?) are examples for this kind of editor. They differ from simple output filter in providing editing functionality for OMDoc specific information.

Human-Oriented Frontend Formats for instance the QMATH project described in Part defines an interface language for a fragment of OMDoc, that is simpler to type by hand, and less verbose than the OMDoc that can be generated by the `qmath` parser. STEX defines a human-oriented format for OMDoc by extending the TEX/LATEX with content markup primitives, so that it can be transformed to OMDoc. See ?projects@stex? for details.

Mathematical Knowledge Bases The MBASE and MAYA systems described in Part II and Part IX are web-based mathematical knowledge bases that offer the infrastructure for a universal, distributed repository of formalized mathematics represented in the OMDoc format.

[1]If your OMDoc project is not represented here, please contact `m.kohlhase@jacobs-university.de` to arrange for inclusion in later editions of this book.

1.2 Application Roles of the OMDoc Format

The applications above support the utilization of the OMDoc format in several roles. Generally, OMDoc can used of as a

Communication Standard between mechanized reasoning systems.

Data Format for Controlled Refinement from informal presentation to formal specification of mathematical objects and theories. Basically, an informal textual presentation can first be marked up, by making its structure explicit (classifying text fragments as definitions, theorems, proofs, linking text, and their relations), and then formalizing the textually given mathematical knowledge in logical formulae (by adding `FMP` elements; see).

Document Preparation Language. The OMDoc format makes the large-scale document- and conceptual structures explicit and facilitates maintenance on this level. Individual documents can be encoded as lightweight narrative structures, which can directly be transformed to e.g. XHTML+MathML or LaTeX, which can in turn be published on the Internet.

Basis for Individualized (Interactive) Documents. Personalized narrative structures can be generated from MBase content making use of the conceptual structure encoded in MBase together with a user model. For instance, the MMiSS, MathDox, and ActiveMath projects described in Part IV to Part VI use the OMDoc infrastructure in an educational setting. They make use of the content-orientation and the explicit structural markup of the mathematical knowledge to generate on the fly specialized learning materials that are adapted to the students prior knowledge, learning goals, and notational tastes.

Interface for Proof Presentation. As the proof part of OMDoc allows small-grained interleaving of formal (`FMP`) and textual (`CMP`) presentations in multiple languages (see e.g. [**HuangFiedler:pvip97**; **Fiedler:uacatp99**]).

Part I

QMath: A Human-Oriented Language and Batch Formatter for OMDoc

[2] EdN:2

QMATH is a batch processor that produces an OMDoc file from a plain UNICODE text document, in a similar way to how TEX produces a DVI file from a plain text source. Its purpose is to allow fast writing of mathematical documents, using plain text and a straightforward syntax (like in computer algebra systems) for mathematical expressions.

The "Q" was intended to mean "quick", since QMATH began in 1998 as an abbreviated notation for MATHML. The first version (0.1) just expanded the formulas found enclosed by "$" signs, which were abbreviated forms of the MATHML element names, and added the extra markup such as `<mrow>` and the like. The second (0.2) did the same thing, but this time allowing an algebraic notation that was fixed in the source code. Finally, version 0.3 allowed the redefinition of symbols while parsing, but it was not capable of expanding formulas embedded in XML documents like the previous ones did until version 0.3.8.[2] For a more detailed history see [**QMathHistory:URL**].

QMATH is very simple: it just parses a text (UTF-8) file according to a user-definable table of symbols, and builds an XML document from that. The symbol definitions are grouped in files called "contexts". The idea is that when you declare a context, its file is loaded and from then on these symbol definitions take precedence over any previous one, thus setting the context for parsing of subsequent expressions.

The grouping of symbols in the context files is arbitrary. However, the ones included with QMATH follow the OPENMATH Content Dictionaries hierarchy so that, for instance, the English language syntax for the symbols in the "arith1" CD is defined in the context "Mathematics/OpenMath/arith1".

Figure 1.1 shows a minimal QMATH document, and the OMDoc document generated from it. The first line ("`QMATH 0.3.8`") in the QMATH document is required for the parser to recognize the file. The lines beginning with "`:`" are metadata items, the first of which, `:en`, declares the primary language for the document, in this case English. Specifying the language is required, as it sets the basic keywords accordingly, and there is no default (privileged) language in QMATH. For example, the English keyword "`Context`" is written "`Contexto`" if the language is Spanish.

[2]EDNOTE: project page: `http://www.matracas.org/qmath/index.en.html`

[2]This offers an alternative to the OQMATH wrapper mentioned in Part VI.

```
QMATH 0.3.8
:en
Context: "Mathematics/OMDoc"

:"Diary"
:W. Smith
:1984−04−04 18:43:00+00:00

Context: "Mathematics/Arithmetic"

Theory:[<−thoughtcrime]

:"Down with Big Brother"
Freedom is the freedom to say $2+2=4$.
If that is granted, all else follows.
```

```
From contexts/en/Mathematics/OpenMath/arith1.qmath:

Symbol: plus OP_PLUS "arith1:plus"
Symbol: + OP_PLUS "arith1:plus"
Symbol: sum APPLICATION "arith1:sum"
Symbol: Σ APPLICATION "arith1:sum"
 ...

From contexts/en/Mathematics/OpenMath/relation1.qmath:

Symbol: = OP_EQ "relation1:eq"
Symbol: neq OP_EQ "relation1:neq"
Symbol: ¬= OP_EQ "relation1:neq"
Symbol: ≠ OP_EQ "relation1:neq"

...
```

```
<?xml version='1.0' encoding='UTF−8' standalone='no'?>
<omdoc xmlns='http://omdoc.org/ns' version='1.6'
       xmlns:dc='http://purl.org/dc/elements/1.1/'>
 <metadata>
  <dc:language>en</dc:language>
  <dc:title>Diary</dc:title>
  <dc:creator role='aut'>W. Smith</dc:creator>
  <dc:date>1994−04−04T18:43:00+00:00</dc:date>
 </metadata>
 <theory xml:id='thoughtcrime'>
  <imports from="arith1"/>
  <imports from="relation1"/>
  <omtext>
   <metadata><dc:title>Down with Big Brother</dc:title></metadata>
   <CMP>
    Freedom is the freedom to say
    <OMOBJ xmlns='http://www.openmath.org/OpenMath'>
     <OMA>
      <OMS cd='relation1' name='eq'/>
      <OMA>
       <OMS cd='arith1' name='plus'/>
       <OMI>2</OMI><OMI>2</OMI>
      </OMA>
      <OMI>4</OMI>
     </OMA>
    </OMOBJ>.
    If that is granted, all else follows.
   </CMP>
  </omtext>
 </theory>
</omdoc>
```

Figure 1.1: A minimal QMath document (top left) and its OMDoc result (bottom). Some symbol definitions are displayed in the top right.

(Similarly, the arithmetic context is "`Matemáticas/Aritmética`"). Then, the "OMDoc" context is loaded, defining the XML elements to be produced by the metadata items and the different kinds of paragraphs: plain text, theorem, definition, proof, example, etc.

After that setup come the document title, author name (one line for each author), and date, which form the content of the OMDoc metadata element.

The document is composed of paragraphs (which can be nested) separated by empty lines, and formulas are written enclosed by "$" signs.

There is an Emacs mode included in the source distribution, that provides syntax highlighting and basic navigation based on element identifiers.

It is also possible to use it on an XML document for expanding only the mathematical expressions. QMath will detect automatically the input format, either QMath text or XML, and in the later case output everything verbatim except for the QMath language fragments found inside the XML processing instructions of the form `<?QMath ... ?>` and the mathematical expressions between "$".

```
<?xml version='1.0' encoding='UTF−8' standalone='no'?>
<?QMath
:en
Context: "Mathematics/Arithmetic"
?>
<omdoc xmlns='http://omdoc.org/ns' version='1.2'
       xmlns:dc='http://purl.org/dc/elements/1.1/'>
 <metadata>
  <dc:language>en</dc:language>
  <dc:title>Diary</dc:title>
  <dc:creator role='aut'>W. Smith</dc:creator>
  <dc:date>1984−04−04T18:43:00</dc:date>
 </metadata>
 <theory xml:id='thoughtcrime'>
  <omtext>
   <metadata><dc:title>Down with Big Brother</dc:title></metadata>
   <CMP>
    Freedom is the freedom to say $2+2=4$.
    If that is granted, all else follows.
   </CMP>
  </omtext>
 </theory>
</omdoc>
```

Figure 1.2: The same example document, using QMath only for the formulas.

While QMath was a good improvement over manual typing of the OMDoc XML, it does not scale well: in real documents, with more than a couple of nesting levels, it is difficult to keep track of where the current paragraph belongs.

One solution is to use it only for the mathematical expressions, and rely on some XML editor for the document navigation and organization, such as the OMDoc mode for Emacs described in or the OQmath mode for jEdit in Part VII. Another is to use the Sentido browser/editor in Part I, which reimplements and extends QMath's functionality.

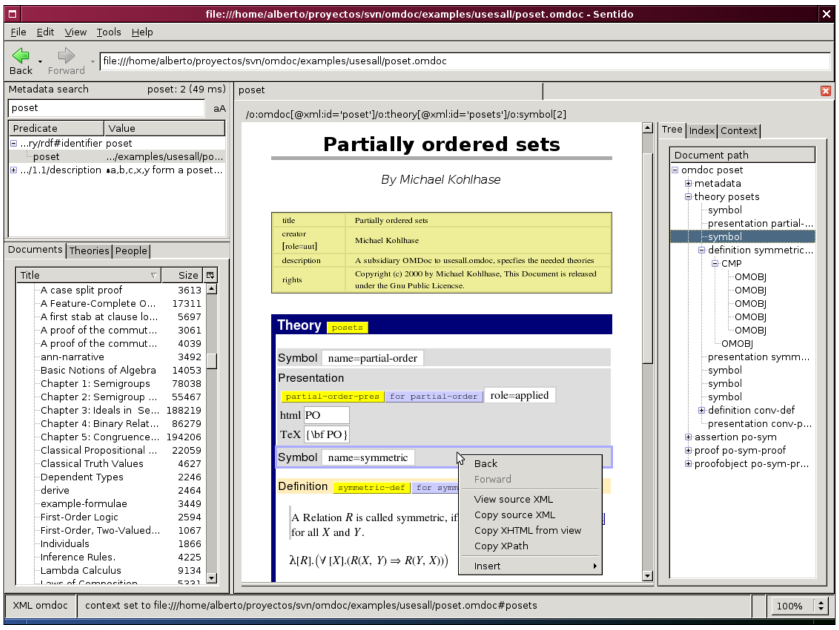
QMath is Free Software distributed under the GNU General Public License (GPL [**GPL**]).

Part II

Sentido: an Integrated Environment for OMDoc

[3] EdN:3

SENTIDO is an integrated environment for browsing, searching, and editing collections of OMDoc documents. It is implemented as an extension for the MOZILLA/FIREFOX browsers to avoid the biggest problems found when using QMATH: the need to compile the program for installing, the batch mode of interaction that made small corrections consume much of the author's time, and the lack of any support for document navigation and search.



Predicate	Value
...ry/rdf#identifier	poset
poset	.../examples/usesall/po...
.../1.1/description	a,b,c,x,y form a poset...

Title	Size
A case split proof	3613
A Feature-Complete O...	17311
A first stab at clause lo...	5697
A proof of the commut...	3061
A proof of the commut...	4039
ann-narrative	3492
Basic Notions of Algebra	14053
Chapter 1: Semigroups	78038
Chapter 2: Semigroup ...	55467
Chapter 3: Ideals in Se...	188219
Chapter 4: Binary Relat...	86279
Chapter 5: Congruence...	194206
Classical Propositional ...	22059
Classical Truth Values	4627
Dependent Types	2246
derive	2464
example-formulae	3449
First-Order Logic	2594
First-Order, Two-Valued...	1067
Individuals	1866
Inference Rules.	4225
Lambda Calculus	9134
Laws of Composition	5331

/o:omdoc[@xml:id='poset']/o:theory[@xml:id='posets']/o:symbol[2]

Partially ordered sets

By Michael Kohlhase

title	Partially ordered sets
creator [role=aut]	Michael Kohlhase
description	A subsidiary OMDoc to usesall.omdoc, specfies the needed theories
rights	Copyright (c) 2000 by Michael Kohlhase, This Document is released under the Gnu Public Licencse.

Theory posets

Symbol name=partial-order

Presentation

partial-order-pres for partial-order role=applied

html PO

TeX {\bf PO}

Symbol name=symmetric

Definition symmetric-def for symm

A Relation R is called symmetric, if
for all X and Y.

$\lambda[R].(\forall [X].(R(X,\ Y) \Rightarrow R(Y,\ X)))$

Document path
- omdoc poset
  - metadata
  - theory posets
    - symbol
    - presentation partial-...
    - symbol
    - definition symmetric...
      - CMP
        - OMOBJ
        - OMOBJ
        - OMOBJ
        - OMOBJ
        - OMOBJ
      - OMOBJ
    - presentation symm...
    - symbol
    - symbol
    - symbol
    - definition conv-def
    - presentation conv-p...
  - assertion po-sym
  - proof po-sym-proof
  - proofobject po-sym-pr...

context set to file:///home/alberto/proyectos/svn/omdoc/examples/usesall/poset.omdoc#posets

Figure 1.3: SENTIDO after indexing the OMDoc repository in the library (left) and loading a document from it (center and right).

[3]EDNOTE: projecthomepage `http://www.matracas.org/sentido/index.en.html`

Figure 1.3 shows a typical session initiated by searching in the document library (described below in more detail) and opening one of the results. The context menu displays the options for browsing back and forward, viewing the XML (OMDoc) source of the selected element, copying it to the system clipboard, copying its MathML rendering or an XPath expression that identifies it, and inserting new elements.

Chapter 2

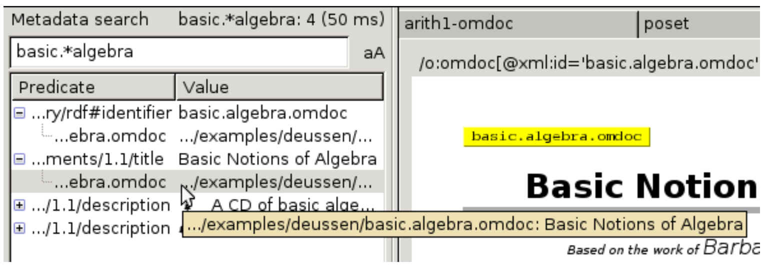
The User Interface

The window is made to resemble the web browser, and consists of two main panes: the smaller one on the left contains the interface for the “document library”, and the right one the “document view” and associated information like the document tree, element identifiers index, and context at the current cursor position.

The document library is a knowledge base about documents, the theories defined in them, and people mentioned in their metadata as authors, editors, translators, etc. It is implemented as an RDF store with the documents organized in collections called “volumes” with references to documents, so that different volumes can have documents in common. The tabs labelled “Documents”, “Theories” and “People” display different views of the library content.

The bibliographic data for each document is stored using the Bibliographic Record Schema [**Lennox04**], which includes FOAF[1] entries for people.

The documents in the library are indexed by the search engine, which stores their metadata entries and theory identifiers in an abridged inverted index to speed up the searches to the point where “search as you type” becomes possible[2]. The search pattern accepts regular expression syntax, as shown in Figure 2.1.



Metadata search basic.*algebra: 4 (50 ms)

basic.*algebra

Predicate	Value
...ry/rdf#identifier	basic.algebra.omdoc
...ebra.omdoc	.../examples/deussen/...
...ments/1.1/title	Basic Notions of Algebra
...ebra.omdoc	.../examples/deussen/...
.../1.1/description	A CD of basic alge...
.../1.1/description	

.../examples/deussen/basic.algebra.omdoc: Basic Notions of Algebra

arith1-omdoc poset

/o:omdoc[@xml:id='basic.algebra.omdoc'

basic.algebra.omdoc

Basic Notion

Based on the work of Barba

Figure 2.1: Metadata Search in Sentido: tooltips show the content of the cropped entries.

The document view is built using XHTML + MathML that can be edited normally, with the changes being propagated to the internal OMDoc.

The view is built on demand (using XSLT) as the subparts of the document are unfolded in the document navigation tree found in the right part of the window. This has been found important in practice since many real uses of OMDoc involve documents that contain large lists of elements, like exercises, that are largely independent of each other and thus do not usually require being viewed at the same time, and the biggest delay in opening a medium to large sized document

[1] “Friend of a friend”, described in their web page `http://www.foaf-project.org` as being “about creating a Web of machine-readable homepages describing people, the links between them and the things they create and do.”

[2] On the author’s 1 GHz laptop computer, the search times in a library of around two thousand documents are usually between 100 and 200 milliseconds.

was by far the display of the XHTML view. Another motivation for this approach is to progress towards handling the source document more like a database, and customize its presentation for the task at hand.

SENTIDO adds some options to the context menu in the browser, to allow the user to open links to OMDoc files from web pages (see Figure 2.2).

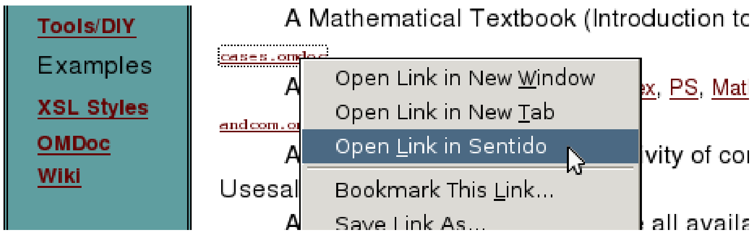


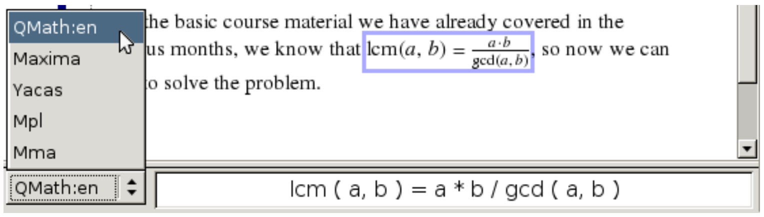
Figure 2.2: MOZILLA's Context Menu after Installing SENTIDO.

Chapter 3

Formula Editing

Mathematical expressions are entered using a selectable linear syntax, translated by a new version of the QMath parser described in Part . This is a much more capable implementation based on finite-state cascades [**abney96partial**].

There are five grammars included in the install package, that are used for translating back and forth between OpenMath and the linear syntax of QMath and the Computer Algebra Systems Maxima, Yacas, Maple™ and Mathematica®. More syntaxes can be added by writing new grammars, with a format similar to QMath "context" files.

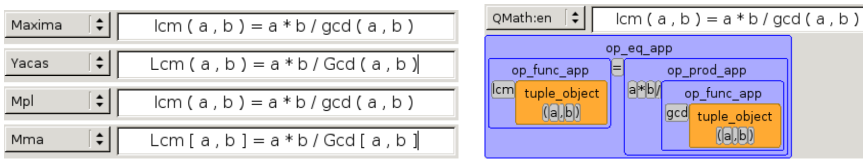


he basic course material we have already covered in the
us months, we know that lcm$(a, b) = \frac{a \cdot b}{\gcd(a, b)}$, so now we can
o solve the problem.

lcm (a, b) = a * b / gcd (a, b)

Figure 3.1: The formula editor under the document view, with the input syntax menu and the text field where the formula is typed, which updates continuously the internal OpenMath representation and the MathML view.

When the cursor enters a formula, the linear input field appears at the bottom of the document view, as seen in Figure 3.1. It contains a text field for editing, and a menu button for selecting the syntax, which can be done at any moment: the linear expression is regenerated instantaneously from its OpenMath form, so it is possible to enter a formula using, for instance, Mathematica® syntax, then select another syntax such as Maple™ , and get the expression immediately translated, going through its OpenMath representation (Figure 3.2).



lcm (a , b) = a * b / gcd (a , b)
Lcm (a , b) = a * b / Gcd (a , b)
lcm (a , b) = a * b / gcd (a , b)
Lcm [a , b] = a * b / Gcd [a , b]
lcm (a , b) = a * b / gcd (a , b)

Figure 3.2: The formula is translated by Sentido each time the user selects another syntax (left, the vertical line is the blinking caret), and it is possible to view the parse tree (right), updated as the input is modified.

Insertion of formulae is achieved by typing the dollar symbol "$", which produces an empty formula readily editable so that the sequence of keystrokes is similar to typing TEX or QMath text: one can type `$e^(pi*i)+1=0$` and get $e^{\pi i}+1=0$ without having to look at the formula editor or use the mouse. The changes as the formula is being modified are stored, and the display updated from the OpenMath form, at each point when there is a complete parse of the formula. This gives immediate feedback on how the program understands the input.

An important difference is that there is no need to care about "context files" any more. In QMath, specifying a "context" had a double function: putting symbols in scope for disambiguation, and selecting a notation style for them. Those aspects are separated in Sentido: the in-scope symbols are automatically determined from the enclosing theory and those imported from it (recursively), and the notation is selectable by the user.

Note that the parser allows any characters supported by the browser rendering engine of Mozilla/FireFox (a big subset of Unicode), not just ASCII. For example, the number 3.14159265... can be entered either as π or with an ASCII form depending on the selected syntax: "pi" for QMath, "%pi" for Maxima, or "Pi" for Yacas, Maple™ and Mathematica®.

Chapter 4

Future Work and Availability

SENTIDO is a long term personal project that has been in development for several years (since 2004), entirely in the author's spare time and using his own computing resources, based on experiments[1] and notes collected during the development of QMATH. Therefore, we expect it to continue developing during the foreseeable future unless a better application appears that makes it redundant.

Its components are designed to be reusable, which is tested from time to time by producing spin-off applications that use subsets of its functionality in a self-contained way. One example is the small Computer Algebra System called ALGEBRA [**algebra:URL**], that contains parts of SENTIDO such as the new parser combined with specific ones like the function plotter and the term rewriting engine.

Future developments will focus on what we consider the two main tasks for a development environment for semantic encoding of mathematical content:

- Ease the tedium of writing all the details needed for an unambiguous encoding of the content. This is where the flexible input parser comes into play: having a syntax redefinable at any point in the content simplifies the expression input, as the syntax can be adapted to the context in which an expression occurs.
- Provide some benefit once we have the semantic encoding which would not be present with an ambiguous encoding such as TEX. Here we need to implement detailed checking and strong search capabilities. A next step would be to assist the writing process by inferring new content and informing the input interface about the context as mentioned above.

Some planned improvements in SENTIDO are:

- Make the browser open OMDoc documents linked from normal pages directly in SENTIDO, by implementing a stream handler for the `MIME` type `application/omdoc+xml`.
- Integrate ALGEBRA into SENTIDO, to add automated symbolic manipulation to the document editing process.
- Extend the checking being done on the theories: at the time of writing these lines, only the theory import relations are checked for loops and unknown theory references, which was already enough to locate several mistyped theory identifiers in the OMDoc repository.
- Implement useful features found in other projects such as THEOREMA [**piroi04environment**]. This is strongly related to the two points above since THEOREMA implements many features needed for the task of content checking which are still missing in SENTIDO, and some of them are available in proof-of-concept form in ALGEBRA.

SENTIDO is Free Software distributed under the GNU General Public License (GPL [**GPL**]).

[1]Some of those early experiments with MOZILLA inspired work done on adapting OPENOFFICE and TEXMACS for OMDoc in collaboration with George Goguadze [**GogPal:amesam03**]

Part III

MBase, an Open Mathematical Knowledge Base

[4] EdN:4

We describe the MBase system, a web-based mathematical knowledge base. It offers the infrastructure for a universal, distributed repository of formalized mathematics. Since it is independent of a particular deduction system and particular logic, the MBase system can be seen as an attempt to revive the Qed initiative [**qed**] from an infrastructure viewpoint. See [**KohFra:rkcimss01**] for the logical issues related to supporting multiple logical languages while keeping a consistent overall semantics. The system is realized as a mathematical service in the MathWeb system [**FraKoh:mabdl99**; **ZimmerMICAI04**], an agent-based implementation of a mathematical software bus for distributed mathematical computation and knowledge sharing. The content language of MBase is OMDoc.

We will start with a description of the system from the implementation point of view (we have described the data model and logical issues in [**KohFra:rkcimss01**]).

The MBase system is realized as a distributed set of MBase servers (see figure 4.1). Each MBase server consists of a Relational Data Base Management System (RDBMS) connected to a mOZart process (yielding a MathWeb service) via a standard data base interface. For browsing the MBase content, any MBase server provides an `http` server (see [**MBase-Demo:URL**] for an example) that dynamically generates presentations based on HTML or XML forms.

This architecture combines the storage facilities of the RDBMS with the flexibility of the concurrent, logic-based programming language Oz [**Smolka:Oz:95**], of which mOZart (see [**Mozart:URL**]) is a distributed implementation. Most importantly for MBase, mOZart offers a mechanism called pickling, which allows for a limited form of persistence: mOZart objects can be efficiently transformed into a so-called pickled form, which is a binary representation of the (possibly cyclic) data structure. This can be stored in a byte-string and efficiently read by the mOZart application effectively restoring the object. This feature makes it possible to represent complex objects (e.g. logical formulae) as Oz data structures, manipulate them in the mOZart engine, but at the same time store them as strings in the RDBMS. Moreover, the availability of "Ozlets" (mOZart functors) gives MBase great flexibility, since the functionality of MBase can be enhanced at run-time by loading remote functors. For instance complex data base queries can be compiled by a specialized MBase client, sent (via the Internet) to the MBase server and applied to the local data e.g. for specialized searching (see [**Duchier:tntb98**] for a related system and the origin of this idea).

[4]EdNote: project page: `http://www.mathweb.org/mbase`

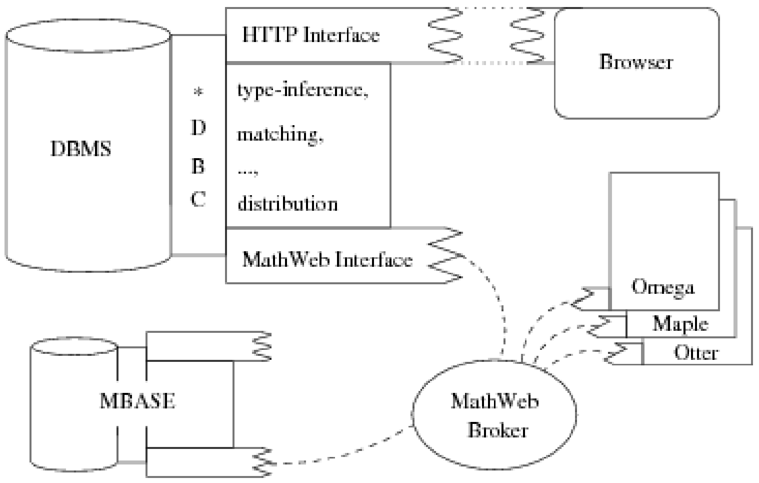


Figure 4.1: System Architecture

MBase supports transparent distribution of data among several MBase servers (see [**KohFra:rkcimss01**] for details). In particular, an object O residing on an MBase server S can refer to (or depend on) an object O' residing on a server S'; a query to O that needs information about O' will be delegated to a suitable query to the server S'. We distinguish two kinds of MBase servers depending on the data they contain: *archive servers* contain data that is referred to by other MBases, and *scratch-pad* MBases that are not referred to. To facilitate caching protocols, MBase forces archive servers to be *conservative*, i.e. only such changes to the data are allowed, that the induced change on the corresponding logical theory is a conservative extension. This requirement is not a grave restriction: in this model errors are corrected by creating new theories (with similar presentations) shadowing the erroneous ones. Note that this restriction does not apply to the non-logical data, such as presentation or description information, or to scratchpad MBases making them ideal repositories for private development of mathematical theories, which can be submitted and moved to archive MBases once they have stabilized.

Part IV

A Search Engine for Mathematical Formulae

[5] EdN:5

As the world of information technology grows, being able to quickly search data of interest becomes one of the most important tasks in any kind of environment, be it academic or not. We present a search engine for mathematical formulae. The MathWebSearch system harvests the web for content representations of formulae (currently MathML and OpenMath) and indexes them with substitution tree indexing, a technique originally developed for accessing intermediate results in automated theorem provers. For querying, we present a generic language extension approach that allows to construct queries by minimally annotating existing representations.

Generally, searching for mathematical formulae is a non-trivial problem — especially if we want to be able to search occurrences of the query term as sub-formulae — for the following reasons:

1. *Mathematical notation is context-dependent*. For instance, binomial coefficients can come in a variety of notations depending on the context: $\binom{n}{k}$, ${}_nC^k$, C^n_k, and C^k_n all mean the same thing: $\frac{n!}{k!(n-k)!}$. In a formula search we would like to retrieve all forms irrespective of the notations.

2. *Identical presentations can stand for multiple distinct mathematical objects*, e.g. an integral expression of the form $\int f(x)dx$ can mean a Riemann Integral, a Lebesgue Integral, or any other of the 10 to 15 known anti-derivative operators. We would like to be able to restrict the search to the particular integral type we are interested in at the moment.

3. *Certain variations of notations are widely considered irrelevant*, for instance $\int f(x)dx$ means the same as $\int f(y)dy$ (modulo α-equivalence), so we would like to find both, even if we only query for one of them.

To solve this formula search problem, we concentrate on *content representations of mathematical formulae*, since they are presentation-independent and disambiguate mathematical notions.

[5]EdNote: project page: `http://search.mathweb.org/`

Chapter 5

A Running Example: The Power of a Signal

A standard use case for MathWebSearch is that of an engineer trying to solve a mathematical problem such as finding the power of a given signal $s(t)$. Of course our engineer is well-versed in signal processing and remembers that a signal's power has something to do with integrating its square, but has forgotten the details of how to compute the necessary integrals. He will therefore call up MathWebSearch to search for something that looks like $\int_?^? s^2(t)dt$ (for the concrete syntax of the query see Listing 7.1 in Chapter 6). MathWebSearch finds a document about Parseval's Theorem, more specifically $\frac{1}{T}\int_0^T s^2(t)dt = \Sigma_{k=-\infty}^{\infty} \mid c_k \mid^2$ where c_k are the Fourier coefficients of the signal. In short, our engineer found the exact formula he was looking for (he had missed the factor in front and the integration limits) and moreover a theorem he may be able to use.

Chapter 6

Indexing Mathematical Formulae

For indexing mathematical formulae on the web, we will interpret them as first-order terms. This allows us to use a technique from automated reasoning called *term indexing* [**Graf:ti96**]. This is the process by which a set of terms is stored in a special purpose data structure (the **index**) where common parts of the terms are potentially shared, so as to minimize access time and storage. The indexing technique we work with is a form of tree-based indexing called *substitution-tree indexing*. A substitution tree, as the name suggests, is simply a tree where substitutions are the nodes. A term is constructed by successively applying substitutions along a path in the tree, the leaves represent the terms stored in the index. Internal nodes of the tree are **generic terms** and represent similarities between terms.

The main advantage of substitution tree indexing is that we only store substitutions, not the actual terms, and this leads to a small memory footprint. Adding data to an existing index is simple and fast, querying the data structure is reduced to performing a walk down the tree. Index building is done in similar fashion to [**Graf:ti96**]. Once the index is built, we keep the actual term instead of the substitution at each node, so we do not have to recompute it with every search. At first glance this may seem to be against the idea of indexing, as we would store all the terms again, not only the substitutions. However, due to the tree-like structure of the terms, we can in fact store only a pool of (sub)terms and define the terms in our index using pointers to elements of the pool (which are simply other terms). To each of the indexed terms, a data string is attached — a string that represents the exact location of the term. We use XPointer [**GroMal:xf03**] to specify this.

Unfortunately, substitution tree indexing does not support subterm search in an elegant fashion, so when adding a term to the index, we add all its subterms as well. This simple trick works well: the increase in index size remains manageable and it greatly simplifies the implementation.

Chapter 7

A Query Language for Content Mathematics

When designing a query language for mathematical formulae, we have to satisfy a couple of conflicting constraints. The language should be content-oriented and familiar, but it should not be specialized to a given content representation format. Our approach to this problem is to use a simple, generic extension mechanism for XML-based representation formats rather than a genuine query language itself. The query extension is very simple, it adds two new attributes to the respective languages: `mq:generic` and `mq:anyorder`, where the prefix `mq:` abbreviates the namespace URI `http://mathweb.org/MathQuery/` for MathWebSearch.

In this way, the user need not master a new representation language, and we can generate queries by copy and paste and then make parts of the formulae generic by simply adding suitable attributes. We will use Content MathML [**CarIon:MathML03**] in the example, but MathWebSearch also supports OpenMath and a shorthand notation that resembles the internal representation we are using for terms (prefix notation). The `mq:generic` attribute takes string values and can be specified for any element in the query, making it into a (named) query variable: its contents are ignored and it matches any term in the search process.

While of searching expressions of the form $A = B$, we might like to find occurrences of $B = A$ as well. At this point the `mq:anyorder` attribute comes in. Inside an `apply` tag, the first child defines the function to be applied; if this child has the attribute `mq:anyorder` defined with the value "yes", the order of the subsequent children is ignored. If we do not want to specify the function name, we can use the `mq:generic` attribute again, but this time for the first child of an `apply` tag. Given the above, the query of our running example has the form presented in Listing 7.1. Note that we do not know the integration limits or whether the formula is complete or not.

Listing 7.1: Query for Signal Power

```
<math xmlns="http://www.w3.org/1998/Math/MathML"
      xmlns:mq="http://mathweb.org/MathQuery">
  <apply><int/>
    <domainofapplication mq:generic="domain"/>
    <bvar> <ci mq:generic="time"/> </bvar>
    <apply><power/>
      <apply><ci mq:generic="fun"></ci><ci mq:generic="time"/></apply>
      <cn>2</cn>
    </apply>
  </apply>
</math>
```

Chapter 8

Input Processing

MathWebSearch can process any kind of XML representation for content mathematics. The system is modular and provides an interface which allows to add XML-to-index-term transformers. . We will discuss input processing for Content-MathML.

Given an XML document, we create an index term for each of its `math` elements. Consider the example on the right: We have the standard mathematical notation of an equation (1), its Content MathML representation (2), and the term we extract for indexing (3). As previously stated, any mathematical construct can be represented in a similar fashion.

1) Mathematical expression:
   $f(x) = y$

3) Term representation:
   $eq(f(x), y)$

2) Content MathML:
```
<apply><eq/>
  <apply>
    <ci>f</ci>
    <ci>x</ci>
  </apply>
  <ci>y</ci>
</apply>
```

Search modulo α-renaming becomes available via a very simple input processing trick: during input processing, we add a `mq:generic` attribute to every bound variable (but with distinct strings for different variables). Therefore in our running example the query variable t (`@time` in Listing 7.1) in the query $\int_?^? s^2(t)dt$ is made generic, therefore the query would also find the variant $\frac{1}{T}\int_0^T s^2(x)dx = \Sigma_{k=-\infty}^{\infty} \mid c_k \mid^2$.

Chapter 9

Result reporting

For a search engine for mathematical formulae we need to augment the set of result items (usually page title, description, and page link) reported to the user for each hit. As typical pages contain multiple formulae, we need to report the exact occurrence of the hit in the page. We do this by supplying an XPointer reference where possible. Concretely, we group all occurrences into one page item that can be expanded on demand and within this we order the groups by number of contained references. For any given result, a detailed view is available. This view shows the exact term that was matched (using Presentation MathML) and the used substitution (a mapping from the query variables specified by the `mq:generic` attributes to certain subterms) to match that specific term.

Chapter 10

Case Studies and Results

We have tested our implementation on the content repository of the ConneXions Project, available via the OAI protocol [**OAI:protocol**]. This gives us a set of over 3,200 articles with mathematical expressions to work on. The number of terms represented in these documents is approximately 53,000 (77,000 including subterms). The average term depth is 3.6 and the maximal one is 14. Typical query execution times on this index are in the range of milliseconds. The search in our running example takes 14 ms for instance. There are, however, complex searches (e.g. using the `mq:anyorder` attribute) that internally call the searching routine multiple times and take up to 200 ms but for realistic examples execution time is below 50 ms. We are currently building an index of the 86,000 Content MathML formulae from `http://functions.wolfram.com`. Here, term depths are much larger (average term depth 5.7, maximally around 50) resulting in a much larger index: it is just short of 2 million formulae. First experiments indicate that search times are largely unchanged by the increase in index size.

In the long run, it would be interesting to interface MathWebSearch with a regular web search engine and create a powerful, specialized, full-feature application. This would resolve the main disadvantage our implementation has – it cannot search for simple text. A simple socket-based search API allows to integrate MathWebSearch into other content-based mathematical software systems.

Part V

Semantic Interrelation and Change Management

[6] EdN:6

The corpus of electronically available mathematical knowledge increases rapidly. Usually, mathematical objects are embedded in and related to different kinds of documents like articles, books, or lecture material, the domain of which can be different from mathematics, e.g., engineering or computer science. Therefore, maintaining high-quality mathematical knowledge becomes a non-trivial engineering task for teams of authors.

In this scenario, sharing and reuse is the key to efficient development. Unfortunately, while there has been a large body of research concerning the sharing and reuse of program developments, sharing and reuse of documents has until now been mainly done by little more than cut and paste. However, to ensure sustainable development, i.e. continuous long-term usability of the contents, sharing and reuse needs to be supported by tools and methods taking into account the semantic structure of the document. In developing these methods and tools we can benefit from the experience in and associated support tools.

We address this problem by providing a methodology to specify coherence and consistency of documents by interrelation of semantic terms and structural entities, supported by a tool for fine-grained version control and configuration management including change management. Semantic interrelation explicates the meaning lying behind the textual content, and relates the semantic concepts within and across documents by means of an ontology. To allow change management, each document is structured in-the-small. Each document corresponds to a package, and packages may be structured in-the-large using folders and import relations. The ideas and methods explained here have been developed in the MMiSS project which aimed at the construction of a multi-media Internet-based adaptive educational system (see [**KriHut:MMiSS03**; **SemInter-DELFI04**; **MMISS-TechReport04**]).

[6]EdNote: project home page: `http://www.mmiss.de`

Chapter 11

Semantic Interrelation Via Ontologies

Ontologies provide the means for establishing a semantic structure. An ontology is a formal explicit description of concepts in a domain of discourse. The MMiSS LaTeX package for ontologies provides a set of easy-to-use macros for the declaration of ontologies in LaTeX documents. They are used to *declare* the ontology of semantic terms used in a document, in a prelude up front. This *specification* of the document contains at least a rigorous hierarchical structure of the terminology (a taxonomy, the *signature* of the document), and may be seen as an elaborate index structure. Moreover, relations between terms may be defined for more semantic interrelation.

The ontology serves a dual purpose — just as the specification of an abstract data type in program development: it specifies the content to be expected in the body of the document in an abstract, yet precise, manner — the content developers requirement specification; and it specifies the content for reference from the outside — the user's perspective, who may then view the body of the document as a black box. The content developer will use the MMiSS LaTeX `Def` command to specify the *defining* occurrence of a promised term, as for an index. Using the structuring in-the-large facilities via packages, the external user may then refer between documents using various kinds of *reference* commands, as the content developer may within a document.

The next section will show, how we can explore this domain ontology — supplied by the author — in order to capture semantic relations between document parts and use these relations for supporting a management of change for mathematical documents.

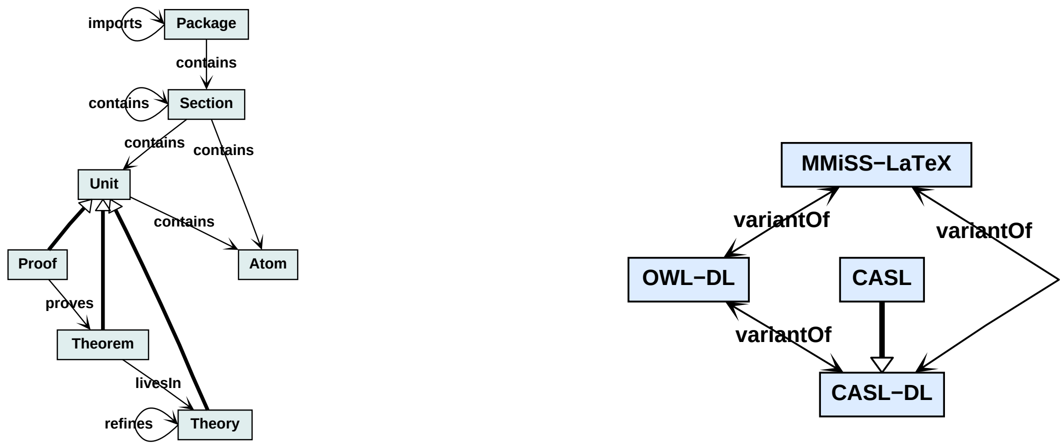


Figure 11.1: (a) Parts of the System's Ontology (b) Formalism variants

Chapter 12

Change Management

The notion of *change management* is used for the maintenance and preservation of consistency and completeness of a development during its evolution. More precisely, we want to have a consistent configuration in which all versions are compatible and there are no cyclic definitions or proofs. At the same time, it should be a complete configuration: there should be no dangling forward references.

Such notions are well-known for formal languages. In contrast, natural language used for writing teaching material does not usually possess a well-defined semantics, and the notion of consistency is arguable. Different authors may postulate different requirements on the material in order to regard it as being consistent. The existence of a user-defined ontology helps a great deal to check references. However, we can make even better use of the information contained in the ontology.

The System's Ontology The aim is to allow change management with regard to consistency and completeness requirements defined by the user in terms of an ontology. In order to unify this approach with the structural consistency and completeness properties introduced above, we express the document structure, originally defined by a document type definition, as an ontology, the so-called *System's Ontology* (see Fig. 11.1a). It defines the following relations between structural elements of documents:

`comprises` An obvious structuring mechanism is nesting of individual parts of a document, leading to the contains relation. The contains relation is part of a family of `comprises` relations that share common properties.

`reliesOn` A family of `reliesOn` relations reflects the various dependencies between different parts of a document. For example, a theorem *lives in* a theory, or proof *proves* a theorem.

`pointsTo` The family of `pointsTo` relations is very similar, and relates references with the defining occurrence of a semantic term.

`variantOf` Another structuring relation is introduced by variants. Parts of a document may e.g. be written in various languages which gives rise to a `variantOf` relation between these document parts and their constituents; it is an equivalence relation.

It is now rather straightforward to formulate consistency and completeness rules in terms of invariants of these relations. Formulating these invariants as formal rules will enable us to implement a generic and flexible change management that keeps track of the invariants and informs the user about violations when a previously consistent document has been revised, leading to various kinds of error (e.g. for `reliesOn` relations) or warning messages (e.g. for `pointsTo` relations).

Properties of Interactions between Structuring Mechanisms. This approach also allows us to lift relations to structuring mechanisms allowing more modular and localized change management. For example, relating the `comprises` and `reliesOn` relations allows us to formalize invariants regarding the closure of document parts with respect to the `reliesOn` relation: We can require that there is a proof for each theorem in a package. Furthermore, if two structural entities are related by `reliesOn`, their relation is propagated along the `comprises` relation towards the root of the hierarchy of nested structural entities, such that (for a theorem T a proof P, and sections A, B):

B contains P & A contains T & P proves $T \Rightarrow B$ `reliesOn` A.

If the user changes section A, the repository will only need to check all sections that A relies on (such as B here) for invariants, and not the whole document. However, in contrast to formal developments as in e.g. the MAYA system [**AH-05-a**], there is no rigorous requirement that a document should obey all the rules. There may be good reasons, for instance, to present first a "light-weight" introduction to all notions introduced in a section before giving the detailed definitions. In this particular case, one would want to introduce forward pointers to the definitions rather than making the definitions rely on the introduction; thus the rules are covered.

In any case, the more structure there is, the better the chances are for preserving consistency and completeness; any investment in introducing more `reliesOn` relations, for example, will pay off eventually. The change management will observe whether revisions by the user will affect these relations and, depending on the user's preferences, emit corresponding warnings.

The aim is to allow users to specify individual notions of consistency by formulating the rules that the relations should obey. This should be possible for the relations between the particular (predefined) structuring mechanisms, but also in general between semantic terms of the user's own ontology. Our work in this direction will rely on the methods and tools provided by the HETS system (see ?projects@hets?).

Chapter 13

Variants

The concept of variants adds a new dimension to hierarchically structured documents. The idea is to maintain and manage different variants of structural entities (document subtrees) which represent the same information in different ways — variants are meant to glue them together.

Managing different natural language variants in parallel is an obvious example. Another one is the formalism variant which denotes the particular formalism in which a formal content part like a theorem or a definition is expressed. Considering ontology development itself, for example, we propose to use variants to maintain different formal representations for the same semantic concept together with its documentation. Figure 11.1b shows the possible variants for declaring ontology components (see [**WOSE-2004**] for details).

The MMiSS repository provides functions to store and retrieve these structural variants by means of specifications for selecting particular variants for editing or presentation.

Chapter 14

Relations to OMDoc

OMDoc provides modules for marking up the knowledge structure and the narrative structure of mathematical documents. MMiSS combines these two viewpoints by giving means for structuring the document contents (which constitutes the narrative structure) and for specifying the incorporated knowledge by use of ontologies. Therefore, we have implemented an export of MMiSS documents to (content and narrative) OMDoc documents and vice versa.

Part VI

MathDox: Mathematical Documents on the Web

[7] EdN:7

The MathDox system provides an infrastructure for interactive mathematical documents that make use of the World Wide Web. These documents take input from various sources, users, and mathematical services. Communication between these different entities can be realized using OpenMath. But, such communication and the interactivity inside the mathematical document take place in a specific, dynamic context. In this paper we discuss our approach to such a dynamic mathematical context: MathDox. It consists of both an XML-based markup language for interactive mathematical contents and a set of software tools realizing the interactivity.

[7] EdNote: project page: `http://www.mathdox.org`

Chapter 15

Introduction

Although the notion of an interactive mathematical document has been around for several years, cf. [**CohMee:tapap98**], its realization is nowhere near the final stage. For instance, recent progress in web technologies has enabled a much smoother communication of mathematics than ever before. The use of an interactive mathematical document (IMD) can provide a window to the world of mathematical services on the Internet, and a mathematical service on the Internet can be created by the building of an interactive mathematical document. MathDox is an ensemble of software tools for creating IMDs, it includes

1. an XML based language that offers markup support for the source texts of IMDs;
2. a document server, rendering interactive mathematical documents from source text and interactively obtained information;
3. mathematical services, providing connections with CASs like Mathematica® and GAP via OpenMath phrasebooks (cf. [**URL:omsoc**]).

The creation of MathDox is a project at the Technische Universiteit Eindhoven (the RIACA institute). Several people at RIACA have helped creating it; here we mention Manfred Riem, Olga Caprotti, Hans Sterk, Henny Wilbrink, Mark Spanbroek, Dorina Jibetean. The system is mainly built with Java and related technology. The products are available via the project web site and will be licensed under the Lesser Gnu Public License [**LGPL**].

Chapter 16

The Language

The MathDox source is an XML document. We have derived our own document type definitions (DTD) for these source texts. We have been influenced by both DocBook [**WalMue:dtdg99**] and OMDoc. The former is a fairly general standard for electronic books, the latter is a very rich, and strongly logic-oriented standard for mathematical documents—the main subject of this book. Both OMDoc and MathDox use OpenMath [**BusCapCar:2oms04**], the difference being that OMDoc focuses on representing mathematical knowledge whereas MathDox focuses on interactivity. The connections with both DocBook and OMDoc are of importance to us because we expect several authoring tools for it to emerge in the coming few years, and we want to profit from their presence.

The mathematics in the MathDox source is given by means of OpenMath objects. This feature has clear advantages in terms of portability. The DocBook type grammar sees to it that there are natural scopes, where mathematical objects 'live'. For instance, when a chapter begins with "Let $\mathbb{F}$ be a field", the scope of the variable $\mathbb{F}$ is assumed to be the whole chapter (although, somewhere further down the hierarchy, say in a section of the chapter, this assignment can be overridden).

Interactivity in MathDox is taken care of by XML tags representing various programming constructs as well as queries to external mathematical services. These actions take place within part of the context, which fixes the precise semantics of the objects involved. Further constructs are available for handling context and user input. Our notion of context is based on [**FraHes:aoidms99**]. Context is divided into static and dynamic context. The static context may be defined as the set of all XML sources from which a interactive document can be prepared for use. Two extreme forms are OpenMath Content Dictionaries and a chapter of an ordinary book. The dynamic context behaves more like the state of a CAS. It keeps track of the variables introduced, their properties, their values, and their scopes. The MathDox language has constructs for storing and changing this information. For example, the field $\mathbb{F}$ introduced at the beginning of a chapter may be specified to be a finite field of five elements in the context of a particular section of the chapter.

Although semantics is the primary target, some features for presentation have been built into the language. In order to have a flexible presentation, we use presentation-annotated OpenMath. In MathDox we allow style attributes inside OpenMath objects. By discarding these style attributes, regular OpenMath is obtained. For instance, by providing the appropriate value for the style attribute, the author has a choice between a slash and a fraction display. In $\frac{3/4+2/3}{5}$ we have used them both.

Another way of solving presentation issues is illustrated by the statement: $3, 4 \in \mathbb{Z}$. The corresponding OpenMath expression would be the equivalent of $3 \in \mathbb{Z} \wedge 4 \in \mathbb{Z}$, but our OpenMath statement reads that the sequence $3, 4$ belongs to $\mathbb{Z}$. So here, the semantics of the element-of symbol has been stretched so as to help out presentation.

Chapter 17

The MathDox System

An essential component of the MathDox software is its document server. It provides a view to the client of the content and manages both the static and the dynamic context. The usage of the MathDox document server is shown in Figure 17.1. We explain in some detail the main components shown in this picture.

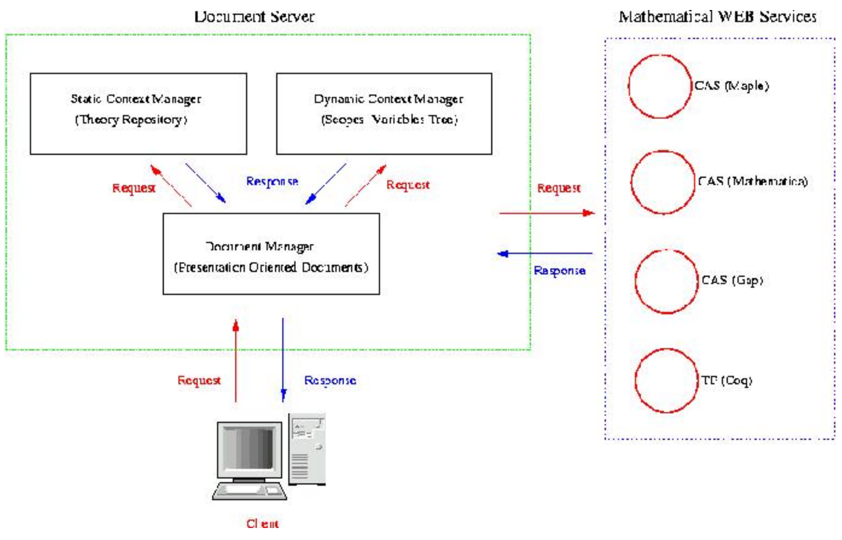


Figure 17.1: The MathDox software

1. The *client*. The client is realized by a math-enabled web browser. It will present views of the documents served to the user, interact with the user, and communicate user input to the document server.

   The communication between client and server takes place via the HTTP request/response mechanism. The responsibility for interaction is mostly on the server side.

2. The *document server*. This server caters for presentation, communication, and context. It supports a wide range of actions ranging from handling queries to searching within documents for mathematical content and from placing (and retrieving) objects into the context, to rendering documents in different views.

   The document server is realized as a Java enhanced web application [**URL:jsp**] inside a web server. It is not a monolithic entity. As shown in Figure 17.1, it is formed by the system parts. The *document manager* serves views to the client. IMDs can be thought of as programs (scripts) encoding the production of a response. In generating the response, they can make use of the information contained in the static context, and in the dynamic context (scopes and variables), the user input communicated along with request, and the results of computations carried on by one or more mathematical services.

Another part is the *static context manager* which is responsible for managing a repository of MathDox mathematical theories.

The final (third) part is the *dynamic context manager* which is responsible for the dynamic information.

3. *mathematical services.* Mathematical services can be very diverse: some may serve as general interfaces to CAS or to Theorem Provers. The MathDox software provides ways to access these services via standard protocols, among which those developed under the MONET project [**URL:monet**]. The mechanism extends the phrasebook set-up for OpenMath [**CapCoh:uosdmc00**; **CapCoh:jpcaad00**]. For constructing specific OpenMath services, we employ our Java OpenMath library ROML [**URL:roml**].

Chapter 18

Conclusion

Now that MathDox is close to a complete working version, trial applications are in the make. We mention

- a server for providing designs of experiments on command to statisticians,
- an exercise repository for the EU funded LeActiveMath project,
- a mathematics course on calculus, with automated natural language production from a formal-mathematical source for the EU funded project WebALT,
- interactive lecture notes (the successor of [**CohCuySterk:ida99**]) for an Abstract Algebra course within a mathematically oriented Bachelor curriculum,
- educational material for highschool mathematics in the Netherlands.

Part VII

OMDoc in ActiveMath

ActiveMath is a mature web-based intelligent learning environment for mathematics that has been developed since 2000 at the University of Saarland and at the German Research Institute of Artificial Intelligence (Intelligent Learning Environments Group headed by Erica Melis). Its learning objects are encoded in an extension of OMDoc.

Chapter 19

The ActiveMath System

In addition to presenting pre-defined interactive materials, it adaptively generates courses according to the learner's goals, learning scenarios, competencies, and preferences. For this, Tutorial Component requests learningobjects [1], related to the learning goal to be retrieved from several repositories. The retrieval of object-IDs is realized by a mediator taking into account structures and meta data of learning objects, and then the Tutorial Component assembles them to a course skeleton depending on a Learner Model. For details see [**Ullrich-TutorialPlanningYRT-AIED-2005**; **Ullrich-InstructionalOntology-ISWC-2004**].

In several stages a Presentation Component fills and transforms this skeleton to a material in the requested output format. In the interactive browser formats dummies can represent Learning Objects that can be instantiated dynamically — depending on the learning progress or on requests by the user.

This Learner Model stores the learning history, the user's profile and preferences, and a set of beliefs that the systems holds about the cognitive and meta-cognitive competencies and the motivational state of the learner. The domain model that underlies the structure of the learner model is inferred from the content for that domain and its meta data represented in the OMDoc source.

ActiveMath is internationalized and 'speaks' German, English, French, Spanish, Russian, and Chinese by now. Its mathematical notation rendering can as well be adapted to national' standards.

To realize a smooth and efficient cooperation of all components and in order to integrate further internal and external services, ActiveMath has adopted a modular service-oriented architecture displayed in Figure 19.1. It includes the XML-RPC web communication protocol for its simplicity and support. In addition, an event framework enables the asynchronous messaging for any changes.

A complex subsystem in its own right is ActiveMath's exercise subsystem [**icce05**] that plays interactive exercises, computes diagnoses and provides feedback to the learner in a highly personalized way. It reports events to inform the other components about the user‘s actions.

In 2005, large educational contents exist in ActiveMath's repositories for Fractions (German), Differential Calculus (German, English, Spanish) at high school and first year university level, operations research (Russian, English), Methods of Optimization (Russian), Statistics and Probability Calculus (German), Matheführerschein (German), and a Calculus course from University of Westminster in London.

19.1 ActiveMath's Service-Approach

The encoding of content in OMDoc is an advantage for ActiveMath's Web-service approach. If available, the services – including Web repositories – can communicate more semantic information

[1] Following the classical definitions, learning objects are any resources that are used the learning activity. When in OMDoc, learning objects considered are such as a `definition`, an `omtext` or an interactive exercise.

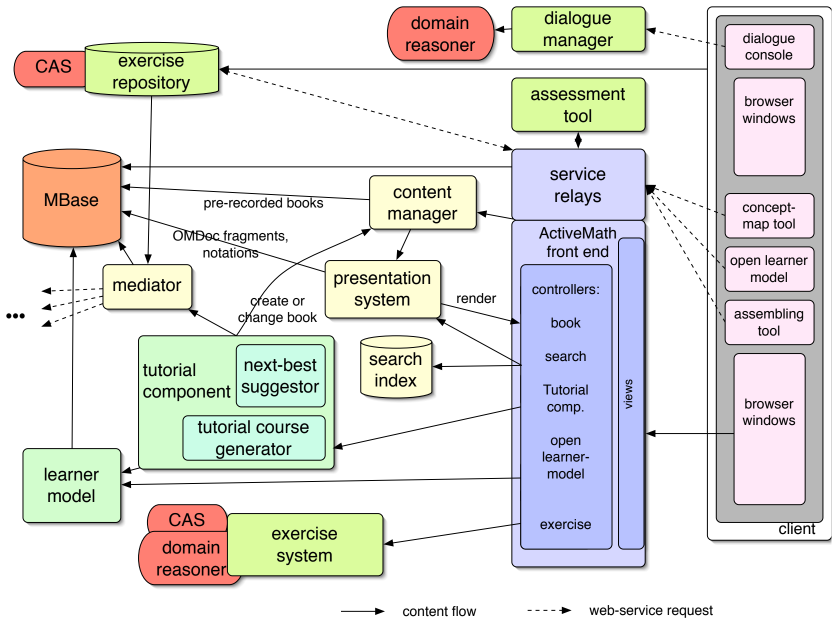


Figure 19.1: The Components, Services and Information Flow in ActiveMath

than just meta data. However, the interoperability of the content encoding is only one side of the Semantic Web coin. Hence, the developments for ActiveMath also include the reuse and interoperability of components and tools [**Melisetal-SemanticAware-BJET-2005**].

External services that are being connected currently are the Siette assessment tool [**Conejo-Siette-IJAIED-04**] and one or more repository of interactive exercises and interactive content.

Chapter 20

OMDoc Extensions for ActiveMath

The ActiveMath DTD extends the OMDoc DTD version 1.1 in several directions:

- new types of items such as `misconceptions`, additional types of items such as types of exercises (`MCQ`, `FIB`, `map`, `problem`),
- additional several relations with types such as `for` or `prerequisite-of`,
- other additional meta data such as `difficulty`, `competency`, or `field`,
- additional infrastructure as, e.g., in exercises, additional structure such as content packages [**LeAMD6**].

The metadata and relation extensions are compliant with the Learning Metadata Standards IEEE and IMS LOM [**lom3˙6**; **ims˙lom**]. Most of the extensions are pedagogically/educationally motivated. Some details follow.

The educational metadata include `competency` and `competencylevel` that are used for assessment, evaluation, and for adaptive suggestions of examples and exercises in course generation. As for competencies, ActiveMath supports Bloom's taxonomy of learning goal levels [**bloom56**] and the more recent taxonomy from the Program for International Student Assessment (PISA) [**klieme04**] and National Council of Teachers of Mathematics (NCTM).

ActiveMath educational metadata include `learning_context` which was in first versions of LOM. Metadata values, such as `difficulty`, `abstractness`, and "typical learning time" have been annotated with the corresponding learning context (allowing to say that an example is hard for an undergraduate but not for a higher class). The ActiveMath DTD introduced some educational relation types which facilitate adaptive course generation and concept map exercises, among others.

The OMDoc format has been refactored in ActiveMath in order to represent metadata in a form that is separable from the representation of the knowledge item. For example, some metadata represented in form of attributes of an item is moved inside the metadata element. The purpose of such a separation is to facilitate the management of learning materials in ActiveMath. Components such as Tutorial Component and Learner Model do not deal with the content of the knowledge items but rather with their metadata only and hence it is convenient to have a way to extract metadata records from the content.

For the internationalization each OMDoc item may have sub-elements in several languages since ActiveMath does not translate learning objects on the fly.

ActiveMath extends the OMDoc `example` element. A detailed explanation can be found in [**Melisetal-FadedEx-ITS04-2004**]. In case of a worked-out example, the micro-structure of this element is enriched with a solution that has a structure similar to a proof in OMDoc.

It differs from the proof element since the solution might not only prove a statement, but also calculate the value of some expression or explore the properties of a particular structure (e.g. curve discussion). This representation allows for different presentations, and serves as a basis for the automatic generation of exercises by fading some parts of the structure of a worked-out example (see [**Melisetal-FadedEx-ITS04-2004**]).

The new exercise representation of ActiveMath was the basis for extending the Math QTI standard [**ENCS04**]. Even though its origin can be traced to OMDoc originally not much is left from the QUIZ representation of OMDoc which supports only very limited types of exercises and did not have enough infrastructure. The micro-structure of an interactive exercise has to allow for different kinds of interactivity, checking the correctness of the answer, providing feedback, etc. This interaction graph can be automatically filled with information by the exercise subsystem components that can communicate with external systems in order to generate feedback to the user.

A description of ActiveMath language for exercises can be found in [**icce05**].

Chapter 21

Usage of Semantic Representation in ActiveMath

The fact that the Tutorial Component employs metadata to search for appropriate learning objects and assemble them has been sketched above. In addition, other tools and components of ACTIVEMATH make use of the semantics of OPENMATH, the ACTIVEMATH metadata and OMDoc more generally.

21.1 Computer Algebra Services

Computer algebra system (CAS) — currently YACAS [**URL:Yacas**], MAXIMA [**URL:Maxima**], and WIRIS [**URL:wiris-cas**] — are integrated as external services. Via a broker, a CAS receives queries (partially Monet queries) to evaluate OPENMATH expressions. This enables the exercise system to evaluate user input, e.g., for numerical or semantic equivalence with a particular expression. The service CAS has to translate in- and output via phrasebooks.

21.2 Presentation Component

The naive approach to rendering OMDoc documents would be to fetch the items from a data base, assemble them (or parts of them) and then run several style-sheets on the resulting sequence; Those style-sheets would depend on the requested output format (HTML + Unicode, XHTML + MATHML, PDF via LaTeX, SVG, or slides), the target browser (we support MOZILLA, FIREFOX, Internet Explorer) and the personalization.

This approach turned out to be infeasible for complex, real-world applications. Therefore ACTIVEMATH includes a multi-stage presentation process as described in [**Ullrichetal-Presentation-ICALT04**]. It has many advantages, among them a much better performance and even better perceived performance through multiple caching, a clear separation of different concerns which provides more flexibility for the various adaptivity dimensions that ACTIVEMATH supports, including selection of learning objects, link annotations language, specific presentations of pages, exercises etc, and of mathematical expressions, target output format, browser.

The final rendering maintains the references to mathematical symbols but renders them invisible. This information can then be used by copy-and-paste and for tool tips that indicate the name of a symbol on the page.

For an even more specialized presentation of mathematical notation which is often requested by authors and users we developed a complex presentation tag representation and an authoring facility for it [**ManLib:apo05**]. These special presentations are integrated into the presentation process upon request.

21.3 Copy and Paste

The rendering includes an invisible reference to the unique identifier of mathematical symbols and expressions. This provides a basis for copying the reference to an OpenMath expression, i.e., the semantics of the expression to a computer algebra system, to the input editor (in dictionary and exercises), and into exercise blanks. The actual transfer mechanism is, because of security limitations and because of resource management, a drag-and-drop operation which allows immediate negotiation between the recipient and source. This allows to transform to the appropriate encoding on demand. Alternate encodings include OpenMath with a restricted set of content-dictionaries, HTML with embedded presentation and content MathML. Reference to OMDoc items and to pages of a book in ActiveMath are exchangeable using the same paradigm.

21.4 Interactive Concept Map Tool icmap

icmap utilizes the OMDoc encoding and relations for generating feedback to users' inputs [**MelisKaergerHomikcma**
The tool visualizes (parts of) a domain and relations between concepts and between concepts and satellites.

21.5 Semantic Search

ActiveMath's search facility has been upgraded to enable not only approximate search results but also to search semantically for (OpenMath) mathematical expressions, for certain types of learning objects and objects with particular metadata. The implementation of the search uses Jakarta Lucene with its high-performance and easy deployment.

21.6 OMDoc-Related Components and Tools of ActiveMath

Many of the tools described above have not been sufficient for the purposes of a complex and mature educational application such as ActiveMath. Therefore, we had to improve them or implement some from scratch. In particular, these include authoring tools (for which improvement is still ongoing), transformation tools, validation tools, and style sheets. Moreover, new tools have been developed or integrated into ActiveMath, e.g., an input editor that returns OpenMath.

The conversion of OMDoc source to presentation code is done using XSLT style sheets. We started with the style sheets available in OMDoc repository and added to them the ActiveMath linking schemata. These style sheets needed more polishing since they were too big and the management of notations was not feasible. Moreover, the TeX oriented style sheets had to be refurbished in order to work well with big documents.

Further tools have been realized within the authoring tools which are covered in Part VII.

Part VIII

Authoring Tools for ActiveMath

[8] EdN:8

The OMDoc content to be delivered by ActiveMath are OMDoc documents with OpenMath formulae. Experience has shown that writing the XML-source by hand is feasible and even preferred if the author wants to follow the evolution of content's structure. It is similar to HTML editing. However, the complexity of XML makes it hard to keep an overview when writing mathematical expressions. Therefore, the OQMath processor has been implemented: it uses QMath for formulae and leaves the rest of the OMDoc written as usual XML.

OQMath has been integrated in a supporting XML-editor, jEdit. This editor provides structural support at writing XML-documents. Authors, even with no XML-knowledge, can easily write valid document jEditOQMath. This package includes, in a one-click installer, QMath, OQMath, jEdit, and Ant-scripts for publication of the content in ActiveMath knowledge bases. These scripts validate the references in the content. These scripts also provide authors with short cycles edit-in-jEditOQMath-and-test-in-ActiveMath. More about jEditOQMath can be seen from `http://www.activemath.org/projects/jEditOQMath` at [**AM-authoring-from-dev-on**]

jEditOQMath provides search facilities as well as contextual drops from items presented in an ActiveMath window. This way the testing of content in the target environment and the authoring experience are bound tighter together, thus making jEditOQMath closer to the WYSIWYG paradigm without being limited to its simple visual incarnation.

To date, more than 10′000 *items* of OMDoc content has been written using these authoring tools in Algebra and Calculus. This experience with authors considerably improved our understanding of what today's authors need and what different classes of authors can cope with.

Among the greatest difficulties of authoring content for ActiveMath was the art of properly choosing mathematical semantic encoding: the mathematical discourse is made of very fine notation rules along with subtle digressions to these rules... formalizing them, as is needed when writing OpenMath or the QMath formulae for them, turns out to often be overwhelming. The usage of the ellipsis in such a formula as $1, \ldots, k, \ldots, n$ is a simple example of semantic encoding challenge. The knowledge organization of OMDoc that makes it possible to define one's own OpenMath symbols has been a key ingredient to facing this challenge.

Among the features most requested by authors, which we have tried to answer as much as possible, are a short edit-and-test cycle and validation facilities taking in account the overall content.

[8] EdNote: project page:`http://www.activemath.org/projects/jEditOQMath`

Chapter 22

Validation Tools

Automated validation of OMDoc content has many facets. XML-validation with a DTD and Schema is a first step. However there are still many structure rules mentioned only as human readable forms in the OMDoc specifications. References between OMDoc items is another important facet which has been answered by ActiveMath knowledge bases and publishing scripts. Experience has proved that ignoring such errors has lead repeatedly to authors complaining about the weirdest behaviours of the overall learning environment. Many other simple validations could be done in order to support the author, for example the validation of a picture embedding, or of fine grained typing of relations (for example, that a definition should only be *for* a symbol).

Further validation tools are being investigated, for example, those tuned to particular pedagogical scenarios.

Chapter 23

Further Authoring Tools for ActiveMath

jEditOQMath clearly remains for users who feel comfortable with source editing. Experience has shown that authors having written HTML or TeX earlier did not find this paradigm problematic. It is, however, a steep learning slope for beginner authors. A more visual component is being worked upon, able to display and edit visually the children of a `CMP`, including formulae.[1] This component, along with forms and summaries for metadata, should provide a visual environment to edit OMDoc content for ActiveMath in a relatively accessible way.

Another area where source editing has shown difficulties is in the process of authoring exercises with many steps... the rich structure of the exercises, along with the non-neglect able space taken by the display of XML-source has challenged several authors, having difficulties to overview such sources as 600 Kb of OQMath source for a single exercise. A web-based visual authoring environment is under work within the ActiveMath group.

[1]More about the component for OMDoc micro-structure can be read from `http://www.activemath.org/projects/OmdocJdomAuthoring/`.

Part IX

SWiM – An OMDoc-based Semantic Wiki

[9] EdN:9

SWiM is a semantic wiki for collaboratively building, editing and browsing a mathematical knowledge base of OMDoc theories. Our long-term objective is to develop a software that facilitates the creation of a shared, public collection of mathematical knowledge and serves work groups of mathematicians as a tool for collaborative development of new theories. Even though the work reported here was initially motivated by solving the MKM author's dilemma [**KohKoh:cdad04**], we contend that the new application area MKM can also contribute to the development of semantic wikis.

Technically, SWiM is based on the semantic wiki engine IkeWiki [**schaffert06:ikewiki**], which was chosen because of its modular design, its rich semantic web infrastructure, its user assistance for annotations, and its orientation towards learning [**schaffert06:learning-with-semantic-wikis**].

[9]EdNote: project page: `http://kwarc.eecs.iu-bremen.de/projects/swim`

Chapter 24

Semantic Wikis

A wiki [**LeuCun01:wikiway**] is a web server application that allows users to browse, create, and edit hyperlinked pages in a web browser, usually using a simple text syntax. In contrast to most content management systems, wiki pages are accessible via an URL containing their title. A new page can be created by linking from an existent page to the page to be created. This link will then lead to an edit form. Usually, anyone is allowed to edit pages on a wiki, but access can be restricted. Other characteristics of wikis include permanent storage of old page versions (with facilities to display differences between two versions and to restore a certain version), notification about recent changes, and full-text search.

Semantic wikis [**voelkel06:semanticwikistateoftheart**; **TolPas06:wikis-semantic-hypermedia**] enhance wikis by Semantic Web technologies, such as RDF [**LasSwi:rdf99**] or ontologies. Usually one page represents one concept from a real-world domain, which has a type, possibly some metadata, and typed links to other concepts. For example, a link from a wiki page about "Life, the Universe and Everything" to another page about Douglas Adams could be typed as "is author of". In terms of RDF, this can be expressed by the following subject–predicate–object triple,

("Douglas Adams", isAuthorOf, "Life, the Universe and Everything")

where the *isAuthorOf* relation would be defined in an ontology. These links are usually displayed in a navigation box next to the page contents. Semantic wikis only deal with wiki text, not with mathematics, though some allow to embed mathematical formulae as presentational-only TEX.

SWIM encourages users to collaborate: Non-mathematicians can collaborate in creating a "Wikipedia of mathematics" by compiling the knowledge available so far, while scientists can collaboratively develop new theories. Users get an immediate reward for many of their contributions: Once they specify the type of a page or relations of one page to another, this information will be displayed in a box of navigation links. We intend to make the data created in SWIM usable for external services by offering an export facility for OMDoc documents and by integrating them into SWIM. Mathematicians developing theories will be assisted to retain an overview of theory dependencies in order not to break them. Social software services will further utilize the semantic information available from the theories and from tracking the user interaction log ("Who did what on which page when?"). User feedback to pages can be extended to social bookmarking, which is "the practice of saving bookmarks [of Internet resources] to a public web site and 'tagging' them with keywords." [**lomas05:social-bookmarking**] The more users tag a certain resource, the higher a social bookmarking service will rank it.

The enhancements of the data model semantic wikis bring along — compared to traditional wikis — are already present in the OMDoc format, so that an OMDoc-based wiki only needs to operationalize their underlying meaning. For example, typed links, which are implemented via an extension to the wiki syntax in SEMANTIC MEDIAWIKI [**voelkel06:semanticwikipedia**] or editable through a separate editor in IKEWIKI [**schaffert06:ikewiki**], are implemented by means of the `for` attribute to OMDoc's elements (e.g. `<example for="#id-of-assertion">`). SWiM makes them editable easily and visualizes them adequately. A semantic wiki targeted at mathematics must ensure that dependencies between concepts are preserved. Results in this area will be interesting for non-mathematical semantic wikis as well, especially when they support higher levels of formalization such as ontologies.

Chapter 25

Design of SWiM

25.1 Concepts and Relations

The smallest unit that can be displayed, edited, linked to, or archived in a wiki is a page. In a semantic wiki, it usually describes one *concept*, including its properties and its relations to other concepts. While standalone OMDoc documents can contain more than one theory, is is important to keep pages small in a wiki to improve the effectivity of usage. Furthermore, usual semantic wikis only store and display metadata and typed links per page; SWiM does too.[1] Users are strongly encouraged to define at most one theory per wiki page and to roll out non-constitutive statements (see) to separate pages, referencing their context theory. As constitutive statements cannot exist without an enclosing theory, but as, on the other hand, we want each wiki page to form a valid document, we introduced a new element `swim:page`, which can be a child of an `omdoc` element and which has the same content model as a `theory` element — in particular, it can hold several theory-constitutive statements and connect them to their context theory. OMDoc's system ontology has been partly coded in OWL-DL and imported to the wiki's RDF store, which is implemented using the Jena Semantic Web Framework for Java [**URL:jena:web**]. Theories as well as statements of any type form concepts, and the most important relations between those concepts are extracted from the OMDoc pages on saving and then stored as RDF triples. These relations include:

- The import relation between theories
- The relation of a statement to its context theory
- The relation of an example to the statement it exemplifies
- The relation of a proof to the assertion it proves

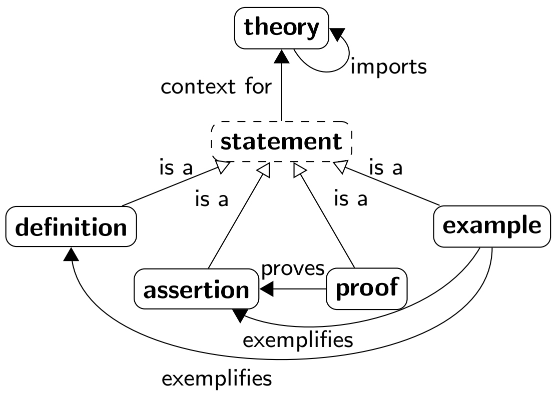


Figure 25.1: Subset of OMDoc's system ontology

It is planned to also take relations given by user interaction into consideration, such as "Who edited which page when?", and to combine ontology-defined relations and user relations. For example, a metric estimating the *degree of difficulty* of a page, calculated by counting the questions on the

[1]Semantic information will only be considered on the theory and statement levels of OMDoc — directly or through reasoning in the case of transitive closures —, not on the object level.

discussion page, could be implemented. Furthermore, the user can specify taxonomic relations, which cannot be stated explicitly in OMDoc, such as ("all differentiable functions are continuous"), as annotations in an ontology language like RDF Schema or OWL.

25.2 User Interface and Interaction Model

Pages can be rendered to XHTML plus presentational MathML using the transformations described in . There is also a browsable source code view, which is useful for documents that are not written in textbook style.

Not only will the user be able to navigate along the dependency graph, she will also be able to *interact* with the system: she will be asked whether she wants to explore the theories required as dependencies in further detail.

Suppose that the user is currently reading the page containing the theory `ring` from the elementary algebra example from . In this case the wiki will not only display navigation links to the direct dependencies `group` and `monoid`, but it will also provide unobtrusive buttons that allow the user to give one of the commands in Figure 25.2. Not only the last case will be recorded — the others are interesting as well for *social bookmarking*. For example, if many users requested a theory t to be explained, the system could default to display not only the direct dependencies but also the level-two dependencies, for it seems that t is too difficult for only being explained shallowly.

No, thanks! "*I already know group and monoid.*"

Explain "*Please show me group and monoid, I want to learn about ring's prerequisites.*" — group and monoid will be displayed.

Explore "*Please show me* all *prerequisites for ring.*" — group, monoid, and semigroup, are opened in separate windows or serialized into one page.

Suspend "*I want to know about group and monoid, but only later.*" — SWIM keeps a notice in the user's profile that she wants to read group and monoid sometime. Reminder links to suspended theories are shown on a separate navigation bar.

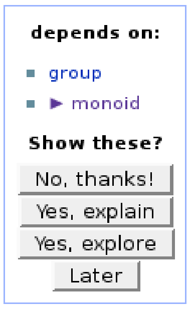


Figure 25.2: The command buttons to navigate along the dependencies

Chapter 26

Further work

Further work on SWiM will concentrate on integrating a lightweight management of change process. Second, while the wiki is yet a user-friendly *browser*, there is still a demand for assisting users to *edit* OMDoc. To this end, the QMath preprocessor (see Part) will be integrated into SWiM. Mathematical objects entered as QMath will be kept in this syntax for display in the edit form, but they will be converted to OMDoc for rendering for presentation and when pages are exported to another application.

Part X

Maya: Maintaining Structured Developments

[10] EdN:10

[10]EdNote: project page: `www.dfki.de/~inka/maya.html`

Chapter 27

Overview

The MAYA-system was originally designed to maintain and utilize the structuring mechanisms incorporated in various specification languages when evolving and verifying formal software developments. In this setting, a software system as well as their requirement specifications are formalised in a textual manner in some specification language like CASL [**CoFI:2004:CASL-RM**] or VSE-SL [**VSE00**]. All these specification languages provide constructs similar to those of OMDoc to structure the textual specifications and thus ease the reuse of components. Exploiting this structure, e.g. by identifying shared components in the system specification and the requirement specification, can result in a drastic reduction of the proof obligations, and hence of the development time which again reduces the overall project costs.

However, the logical formalisation of software systems is error-prone. Since even the verification of small-sized industrial developments requires several person months, specification errors revealed in late verification phases pose an incalculable risk for the overall project costs. An *evolutionary formal development* approach is absolutely indispensable. In all applications so far development steps turned out to be flawed and errors had to be corrected. The search for formally correct software and the corresponding proofs is more like a *formal reflection* of partial developments rather than just a way to assure and prove more or less evident facts.

The MAYA-system supports an evolutionary formal development since it allows users to specify and verify developments in a structured manner, incorporates a uniform mechanism for verification *in-the-large* to exploit the structure of the specification, and maintains the verification work already done when changing the specification. MAYA relies on *development graphs* [**AH-05-a**; **Hutter:mocsv00**] as a uniform (and institution independent[1]) representation of structured specifications, and which provide the logical basis for the *Complex theories* and *Development graphs* of OMDoc[2]. Relying on development graphs enables MAYA to support the use of various (structured) specification languages like OMDoc, CASL [**CoFI:2004:CASL-RM**], and VSE-SL [**VSE00**] to formalise mathematical theories or formal software developments. To this end MAYA provides a generic interface to plug in additional parsers for the support of other specification languages. Moreover, MAYA allows the integration of different theorem provers to deal with the actual proof obligations arising from the specification, i.e. to perform verification *in-the-small*.

Textual specifications are translated into a structured logical representation called a development graph, which is based on the notions of consequence relations and morphisms and makes arising proof obligations explicit. The user can tackle these proof obligations with the help of theorem provers connected to MAYA like Isabelle [**Paulson:iagtp94**] or INKA [**INKA5**].

A failure to prove one of these obligations usually gives rise to modifications of the underlying specification. MAYA supports this evolutionary process as it calculates minimal changes to the logical representation readjusting it to a modified specification while preserving as much verification work as possible. If necessary it also adjusts the database of the interconnected theorem prover. Furthermore, MAYA communicates explicit information how the axiomatization has changed and

[1]This includes, for instance, that it does not require a particular logic (see e.g. [**MAH-06-a**] for more details).
[2]These are the modules CTH and DG, respectively.

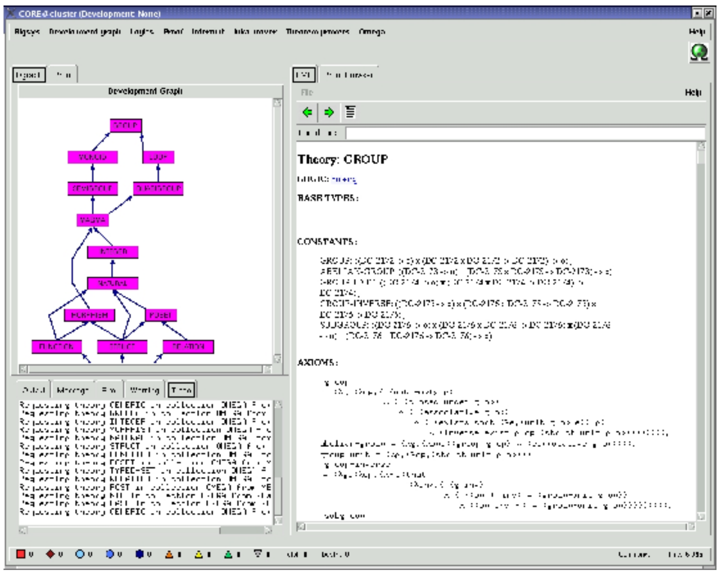
also makes available proofs of the same problem (invalidated by the changes) to allow for a reuse of proofs inside the theorem provers. In turn, information about a proof provided by the theorem provers is used to optimise the maintenance of the proof during the evolutionary development process.

Chapter 28

From Textual to Logical Representation

The specification of a formal development in MAYA is always done in a textual way using specification languages like CASL , OMDoc or VSE-SL. MAYA incorporates parsers to translate such specifications into the MAYA-internal specification language DGRL ("Development Graph Representation Language"). DGRL provides a simply-typed λ-calculus to specify the local axiomatization of a theory in a higher-order logic. While unstructured specifications are solely represented as a signature together with a set of logical formulas, the structuring operations of the specification languages are translated into the structure of a development graph. Each node of this graph corresponds to a theory. The axiomatization of this theory is split into a local part which is attached to the node as a set of higher-order formulas and into global parts, denoted by ingoing definition links, which import the axiomatization of other nodes via some consequence morphisms (such as the `imports` element in OMDoc). While a *local* link imports only the local part of the axiomatization of the source node of a link, *global* links are used to import the entire axiomatization of a source node (including all the imported axiomatization of other nodes). In the same way local and global *theorem link*s are used to postulate relations between nodes (see [**AH-05-a**] for details) which correspond to OMDoc's `theory-inclusion` and `axiom-inclusion` elements.

On the left hand side, Figure 28.1 shows the graphical user interface of MAYA. The right hand side shows the development graph in MAYA for a formalisation of groups. The formalisation was given in OMDoc and imported into MAYA from the OMDoc database MBASE (see [**KohFra:rkcimss01**] and Part II).



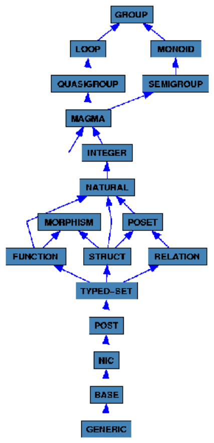


Figure 28.1: The graphical user interface of MAYA & the development graph for the OMDoc Representation of Groups in MBASE

Chapter 29

Verification In-the-large

The development graph is the central data-structure to store and maintain the formal (structured) specification, the arising proof obligations and the status of the corresponding verification effort (proofs) during a formal development.

Maya distinguishes between proof obligations postulating properties between different theories (like the `theory-inclusion` and `axiom-inclusion` elements in OMDoc) and lemmata postulated within a single theory (like `assertion` in OMDoc). As theories correspond to subgraphs within the development graph, a relation between different theories, represented by a global theorem link, corresponds to a relation between two subgraphs. Each change of these subgraphs can affect this relation and would invalidate previous proofs of this relation. Therefore, Maya decomposes relations between different theories into individual relations between the local axiomatization of a node and a theory (denoted by a local theorem link). Each of these relations decomposes again into a set of proof obligations postulating that each local axiom of the node is a theorem in the target theory with respect to the morphism attached to the link.

While definition links establish relations between theories, theorem links denote lemmata postulated about such relations. Thus, the reachability between two nodes establishes a formal relation between the connected nodes (i.e. the theory of the source node is part of the theory of the target node wrt. the morphisms attached to the connecting links). Maya uses this property to prove relations between theories by searching for paths between the corresponding nodes (instead of decomposing the corresponding proof obligation in the first place).

Chapter 30

Verification In-the-small

When verifying a local theorem link or proving speculated lemmata, the conjectures have to be tackled by some interconnected theorem prover. In both cases the proofs are done *within* a specific theory. Thus, conceptually each theory may include its own theorem prover. In principle, there is a large variety of integration types. The tightest integration consists of having a theorem prover for each node wrt. which theory conjectures must be proven, and the theorem prover returns a proof object generated during the proof of a conjecture. Those are stored together with the conjecture and can be used by MAYA to establish the validity of the conjecture if the specification is changed. The loosest integration consists in having a single generic theorem prover, which is requested to prove a conjecture within some theory and is provided with the axiomatization of this theory. The theorem prover only returns whether it could prove a conjecture or not, without any information about axioms used during the proof. For a detailed discussion of the advantages and drawbacks of the different integration scenarios see [**AutMos:ihdgmm02**].

Currently, MAYA supports two integration types: One where information about used axioms is provided by the theorem prover, and one where no such information is provided. In the first case, MAYA stores the proof information and the axioms used during the proof. In the second case, MAYA assumes there is a proof for the proof obligation, as there is no information about the proof. In both scenarios, MAYA makes use of generic theorem provers which are provided with the axiomatization of the current theory. Currently MAYA provides all axioms and lemmata located at theories that are imported from the actual theory by definition links to the prover. Switching between different proof obligations may cause a change of the current underlying theory and thus a change of the underlying axiomatization. MAYA provides a generic interface to plug in theorem provers (based on an XML-RPC protocol) that allows for an incremental update of the database of the prover.

Chapter 31

Evolution of Developments

The user executes changes to specifications in their textual representation. Parsing a modified specification results in a modified Dgrl-specification. In order to support a management of change, Maya computes the differences of both Dgrl-specifications and compiles them into a sequence of basic operations in order to transform the development graph corresponding to the original Dgrl-specification to a new one corresponding to the modified Dgrl-specification. Examples of such basic operations are the insertion or deletion of a node or a link, the change of the annotated morphism of a link, or the change of the local axiomatization of a node. As there is currently no optimal solution to the problem of computing differences between two specifications, Maya uses heuristics based on names and types of individual objects to guide the process of mapping corresponding parts of old and new specification. Since the differences of two specifications are computed on the basis of the internal Dgrl-representation, new specification languages can easily be incorporated into Maya by providing a parser for this language and a translator into Dgrl.

The development graph is always synthesised or manipulated with the help of the previously mentioned basic operations (insertion/deletion/change of nodes/links/axiomatization) and Maya incorporates sophisticated techniques to analyse how these operations will affect proof obligations or proofs stored within the development graph. They incorporate a notion of monotonicity of theories and morphisms, and take into account the sequence in which objects are inserted into the development graph. Furthermore, the information about the decomposition and subsumption of global theorem links obtained during the verification *in-the-large* is explicitly maintained and exploited to adjust them once the development graph is altered. Finally, the knowledge about proofs, e.g. the used axioms, provided by the interconnected theorem provers during the verification *in-the-small* is used to preserve or invalidate the proofs.

Chapter 32

Conclusion and System Availability

The MAYA-system is mostly implemented in Common Lisp while parts of the GUI, shared with the ΩMEGA-system [**SieHes:loui99**], are written in Mozart. The CASL-parser is provided by the CoFI-group in Bremen (see ?projects@hets?). The MAYA-system is available from the MAYA-web-page at `http://www.dfki.de/~inka/maya.html`.

The Heterogeneous Tool Set (HETS, see ?projects@hets?) extends MAYA with a treatment of hiding [**MAH-06-a**], and a uniform treatment of different logics based on the notion of heterogeneous development graphs [**Mossakowski:tdghb02**]. Furthermore, it is planned to extend this with the maintenance of theory-specific control information for theorem provers. The latter comprises a management for building up the database of theorem provers by demand rather than providing all available axioms and lemmata at once and it comprise the management of meta-level information, like tactics or proof plans, inside MAYA.

Part XI

Hets: The Heterogeneous Tool Set

[11] EdN:11

[11]EdNote: project home page: `www.tzi.de/cofi/hets`

Chapter 33

Motivation

"There is a population explosion among the logical systems used in computer science. Examples include first order logic, equational logic, Horn clause logic, higher order logic, infinitary logic, dynamic logic, intuitionistic logic, order-sorted logic, and temporal logic; moreover, there is a tendency for each theorem prover to have its own idiosyncratic logical system. We introduce the concept of institution *to formalize the informal notion of 'logical system'."* [**GoguenBurstall92**]

In the area of formal specification and logics used in computer science, numerous logics are in use:

- logics for specification of data types,
- process calculi and logics for the description of concurrent and reactive behaviour,
- logics for specifying security requirements and policies,
- logics for reasoning about space and time,
- description logics for knowledge bases in artificial intelligence and for the Semantic Web,
- logics capturing the control of name spaces and administrative domains (e.g. the ambient calculus), etc.

Indeed, at present, it is not imaginable that a combination of all these (and other) logics would be feasible or even desirable — even if it existed, the combined formalism would lack manageability, if not become inconsistent. Often, even if a combined logic exists, for efficiency reasons, it is desirable to single out sublogics and study translations between these (cf. e.g. [**Schneider04**]). Moreover, the occasional use of a more complex formalism should not destroy the benefits of *mainly* using a simpler formalism.

This means that for the specification of large systems, heterogeneous multi-logic specifications are needed, since complex problems have different aspects that are best specified in different logics. Moreover, heterogeneous specifications additionally have the benefit that different approaches being developed at different sites can be related, i.e. there is a formal interoperability among languages and tools. In many cases, specialized languages and tools often have their strengths in particular aspects. Using heterogeneous specification, these strengths can be combined with comparably small effort.

OMDoc deliberately refrains from a full formalization of mathematical knowledge: it gains its flexibility through avoiding the specification of a formal semantics of the logic(s) involved. By contrast, the Heterogeneous Tool Set (Hets, [**hets06**]) is based on a rigorous formal semantics. Hets gains its flexibility by providing *formal interoperability*, i.e. integration of different formalisms on a clear semantic basis. Hence, Hets is a both flexible, multi-lateral *and* formal (i.e. based on a mathematical semantics) integration tool. Unlike other tools, it treats logic translations (e.g. codings between logics) as first-class citizens.

Chapter 34

Institutions, Entailment Systems and Logics

Heterogeneous specification is based on individual (homogeneous) logics and logic translations [**MossakowskiHabil**]. To be definite, the terms 'logic' and 'logic translation' need to be formalized in a precise mathematical sense. We here use the notions of *institution* [**GoguenBurstall92**] and *entailment system* [**Meseguer89**], and of *comorphism* [**GoguenRosu02**] between these.

Logical theories are usually formulated over some (user-defined) vocabulary, hence it is assumed that an institution provides a notion of *signature*. Especially for modular specification, it is important to be able to relate signatures, which is done by *signature morphisms*. These can be composed, and hence form a *category of signatures and signature morphisms*.

Furthermore, an institution provides notions of *sentences* and *models* (over a given signature Σ). Models and sentences are related by a *satisfaction relation*, which determines when a given sentence holds in a model. An entailment system also provides an *entailment (provability) relation*, which allows to infer sentences (conclusions) from given sets of sentences (premises, or axioms).

Finally, it is assumed that each signature morphism leads to translations of sentences and models that preserve satisfaction and entailment.

A *institution comorphism* is a translation between two institutions. It maps signatures to signatures, sentences to sentences and models to models, such that satisfaction is preserved (where models are mapped contravariantly, i.e. against the direction of the comorphism).

We refer the reader to the literature [**GoguenBurstall92**; **Meseguer89**; **MossakowskiEA05c**] for formal details of institutions and comorphisms. Subsequently, we use the terms "institution" and "logic" interchangeably, as well as the terms "institution comorphism" and "logic translation".

Chapter 35

The Architecture of the Hets System

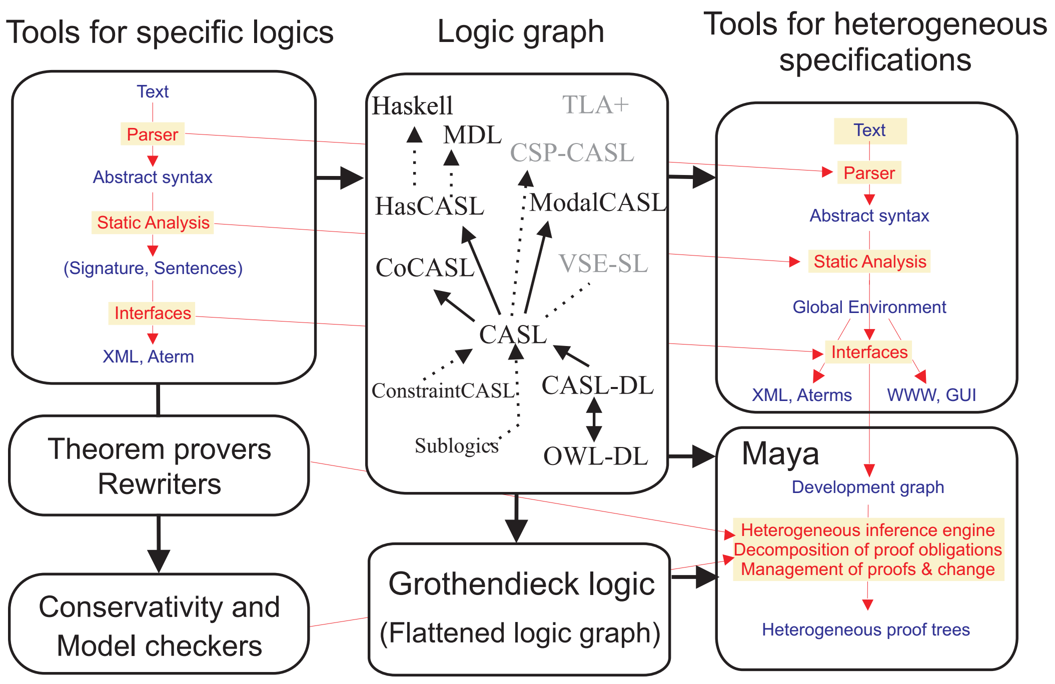


Figure 35.1: Architecture of the heterogeneous tool set

Hets is a tool for parsing, static analysis and proof management combining various such tools for individual specification languages, thus providing a tool for heterogeneous multi-logic specification (see Fig. 35.1). The graph of currently supported logics and logic translations is shown in Fig. 35.2. However, syntax and semantics of heterogeneous specifications as well as their implementation in Hets is parametrized over an arbitrary such logic graph. Indeed, the Hets modules implementing the logic graph can be compiled independently of the Hets modules implementing heterogeneous specification, and this separation of concerns is essential to keep the tool manageable from a software engineering point of view.

Heterogeneous Casl (HetCasl; see [**Mossakowski04**]) includes the structuring constructs of

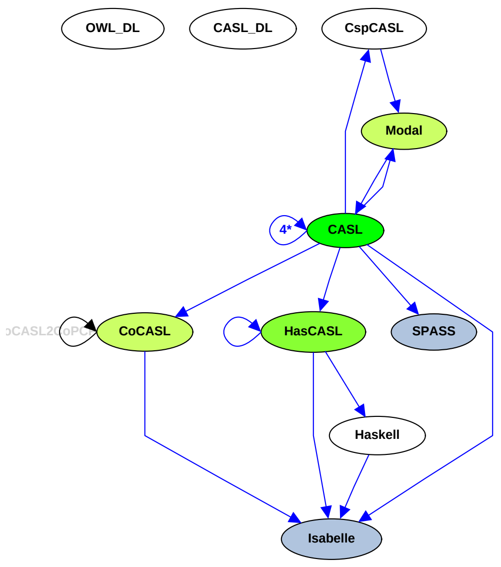


Figure 35.2: Graph of logics currently supported by HETS. The more an ellipse is filled, the more stable is the implementation of the logic.

CASL, such as union and translation. A key feature of CASL is that syntax and semantics of these constructs are formulated over an arbitrary institution (i.e. also for institutions that are possibly completely different from first-order logic resp. the CASL institution). HETCASL extends this with constructs for the translation of specifications along logic translations.

Like MAYA (see Part IX), HETS provides a representation of structured specifications which are the logical basis for the *Complex theories* and *Development graphs* of OMDoc[1].

For proof management, MAYA's calculus of development graphs has been extended with hiding and adapted to heterogeneous specification. Development graphs provide an overview of the (heterogeneous) specification module hierarchy and the current proof state, and thus may be used for monitoring the overall correctness of a heterogeneous development.

HETS also provides a translation of CASL to and from a subset of OMDoc (namely some formal first-order subset). Future work aims at a deeper integration of HETS and OMDoc that provides a translation to and from OMDoc for each of the logics integrated in HETS. Moreover, OMDoc itself will become a "logic" (but only with syntax, without model theory) within HETS, such that also informal OMDoc documents (or formal OMDoc documents written in a logic currently not available in HETS) will be manageable for HETS. In this way, the data formats of OMDoc and HETS will converge, such that tools e.g. for searching, versioning or management of change can be implemented uniformly for both.

[1]These are the modules CTH and DG, respectively.

Part XII

CPoint: An OMDoc Editor in MS PowerPoint

[12] EdN:12

CPoint is an invasive, semantic OMDoc editor in MS PowerPoint (with an OMDoc outlet) that enables a user to distinguish between form and content in a document. As such it can be viewed as an authoring tool for OMDoc documents with a focus on their presentational potential. It enables a user to make implicit knowledge explicit. Moreover, it provides several added-value services to a content author in order to alleviate the short term costs of semantic mark up in contrast to its long term gains.

[12]EdNote: project page `http://kwarc.info/software/CPoint/`

Chapter 36

The CPoint Approach

CPoint started out as a part of the Course Capsules Project (CCaps) at Carnegie Mellon University (2001 — 2004), has been subsequently supported by the International University Bremen (2004) and is now developed further at the 'Digital Media in Education' group at Bremen University. **C**Point is distributed under the Gnu Lesser General Public License (LGPL) [**LGPL**]. The newest version can be downloaded from the project homepage.

PowerPoint (PPT) slides address exclusively the issue of presentation — the placement of text, symbols, and images on the screen, carefully sequenced and possibly animated or embellished by sound. This directly leads to the question: *What exactly is the content in a PPT presentation?*

Obviously, the text and the pictures carry content as does the textual, presentational, and placeholder structure. For instance the ordering of information by writing it in list form, grouping information bubbles in one slide, or marking text as title by putting it into a 'title' placeholder can be mapped directly onto the OMDoc `omgroup` and `metadata` elements. Unfortunately though, this content exploits neither OMDoc's theory level nor the statement or formula level in more than a very superficial way.

The 'real' content is hidden beneath the presentation form: the authors, lecturers, and audience know or learn this real content by **categorizing** what they see, and **combining** it with what they already know and presently hear. **C**Point stands for 'Content in PowerPoint'. It models this by providing the author with a tool to explicitly store the additional implicit knowledge with the PPT show itself and from within the PPT environment without destroying the presentational aspects of the PPT document. Moreover, **C**Point **converts** the additional content to the appropriate OMDoc levels, so that the resulting OMDoc document captures all content. For an author the semantic markup process is a long-term investment. In order to alleviate the author's costs, **C**Point has implemented several **added-value services**.

Chapter 37

The CPoint Application

CPoint extends PPT's presentational functionalities by semantic ones to get a handle on its visible and invisible content. As an invasive editor (see [**Kohlhase:OvercomingProprietaryHurdles**]) CPoint makes these semantic authoring tools available through a toolbar in the PPT menu (see Figure **??**) where they are available whenever PPT is running. CPoint is written in Visual Basic for Applications and can be distributed as a PPT add-in.

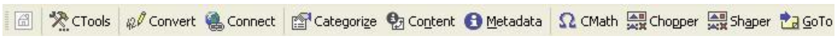


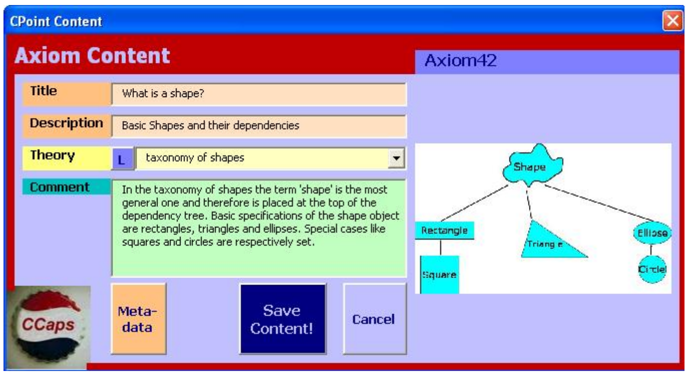
Figure 37.1: The CPoint Menu Bar

The top-level structure of a PPT presentation is given by slides. Each slide contains **PPT objects**, e.g. text boxes, shapes, images, or tables. These objects carry certain properties like text structure (e.g. ordered lists), document structure (e.g. being a title in the text hierarchy), or presentational structure (e.g. color, bold font, italic font, or symbol font). CPoint enables the author to attach additional information to each PPT object. In particular, the author is empowered to transform implicit into explicit knowledge by categorizing, combining and enhancing these objects semantically.

Categorizing The semantic annotation process typically starts with understanding an object's role in the to be transmitted knowledge and a subsequent categorization. The author selects the respective PPT object and assigns a suitable (didactic) role and category from a pre-defined list ranging from hard core mathematical categories like "Theory", "Definition", or "Assertion" to didactic elements like "Question" or "Comment". If a PPT object is part of a multi-part presentation (e.g. ranging over multiple slides) of a semantic entity, it can be marked as a sequel and inherits all information from previous parts. This way the PPT dependant linearity of the objects can be overcome.

Combining For categorized PPT objects the author can input category specific content via the respective details form (see Figure **??** as an example for a PPT group categorized as "Axiom"). In particular, PPT objects can be assigned a relation via CPoint's reference system. For instance, the axiom in Figure **??** sits in the theory called 'taxonomy of shapes'. A more sophisticated example would be a proof *for* an assertion that is constructed out of several, individual proof steps succeeding one another. Frequently, an author wants to reference implicit knowledge (e.g. theories can comprise entire concepts and as such are typically not explicitly presented in a lecture). Here, she can use CPoint to create abstract PPT objects called **abstract objects** that are invisible in the actual PPT show but can be dealt with like all other PPT objects.

The information annotated in these processes can be exploited for added-value services.



Title	What is a shape?
Description	Basic Shapes and their dependencies
Theory	L taxonomy of shapes
Comment	In the taxonomy of shapes the term 'shape' is the most general one and therefore is placed at the top of the dependency tree. Basic specifications of the shape object are rectangles, triangles and ellipses. Special cases like squares and circles are respectively set.

Figure 37.2: The **C**Point Content Form for an Axiom Object

OMDoc Conversion The heart of **C**Point is the functionality for converting a fully (**C**Point-)edited presentation into a valid OMDoc document. This generated OMDoc document can for instance be read into computer-supported education systems like ActiveMath (see [**activemathAIEDJ01**] and Part VI).

Added-Value Services As author support is essential for the motivation doing the semantic markup process, **C**Point offers the following added-value services:

Content Search and Navigation **C**Point's GoTo facility makes use of the additional semantic quality of PPT objects by offering content search. For instance if an author remembers the existence of a definition of "equivalence" in some (older) PPT presentation, she might look up all PPT objects in a collection of several PPT presentations that are categorized as "Definition" and whose title contain the word "equivalence". The author is offered a list of all these objects and by selecting one she is directed to the specific PPT object.

Dependency Graphs CPointGraphs enables the user to view graph based presentations of the annotated knowledge on distinct detail levels.

Semantics-Induced Presentation The module CPointAuthor offers the presentation of the underlying semantics. Whenever the author selects a PPT object basic semantic information (like category, title, and main references) is presented to her. With **C**Point's Visualize Mode semantic labels for annotated PPT objects are generated.

Creation of Pre-Categorized PPT Objects Based on an individually designed CSS style sheet categorized, styled PPT objects can be *created* with CPointAuthor. The layout is determined in the CSS file by the respective category (e.g. proposition) or superordinate classification (e.g. assertion, content, general).

Math Glyphs in PPT Based on the PPT add-in TexPoint, the CMath functionalities empower an author to define individual symbol presentations. **C**Point introduces a mathematical user interface, which fully integrates mathematical symbols into PowerPoint presentations based on the semantics of the underlying objects rather than simply generating appropriate ink marks. For instance, the author might categorize a PPT object as a symbol with the name 'reals' for the real numbers. The specific Unicode character to represent the real numbers can be declared with **C**Point. Subsequently, whenever the author writes the text '`\reals`' and activates the math mode, then this sequence of characters is replaced by the previously declared presentation. The symbol presentation may also be given in LaTeX form so that TexPoint can transform the LaTeX code into PPT glyphs. Note that this feature is not limited to math glyphs but can be used for handy abbreviations (macros) as well.

Editorial Notes Treating PPT presentations as content documents requires more editing, therefore CPointNotes add editorial functionalities like grouped editorial notes and navigation within these.

OMDoc To PPT The CPointImport module enables the import of OMDoc documents into the PPT application. According to an individual underlying CSS style sheet PPT objects in a newly created PPT presentation are generated.

ActiveMath Integrated development environment for ActiveMath content and specific ActiveMath book creation for a selected PPT object.

Chapter 38

Future Work

In the future the addition of other added-value services for users is planned. We want to shift the focus from the authoring role to the recipient role of a PPT presentation, e.g. in form of a CPointStudent module in accordance with the CPointAuthor module. Furthermore, a new, more basic and therefore more user-friendly interface for **C**Point novices will be implemented. This CPointBasic module will try to overcome the heavily form-oriented format of **C**Point. In a next step the growing of a **C**Point user will be supported by offering advanced **C**Point utilities that will extend CPointBasic. Additionally, the success of “social software” under the Web 2.0 paradigm like “social bookmarking” gives rise to the idea of a new personal and sharable PPT objects management where the predefined categories in **C**Point are replaced by “social tags”. Another **C**Point project is its extension for usage by teachers in school, which usefulness has already been established in [**Kohlhase:emPowerPoint**]. The newest project at the International University of Bremen is the implementation of a **C**Point-like editor for MS Word.

Part XIII

STEX: A LATEX-Based Workflow for OMDoc

One of the reasons why OMDoc has not been widely employed for representing mathematics on the web and in scientific publications, may be that the technical communities that need high-quality methods for publishing mathematics already have an established method which yields excellent results — the TEX/LATEX system. A large part of mathematical knowledge is prepared in the form of TEX/LATEX documents.

We present STEX (Semantic TEX) a collection of macro packages for TEX/LATEX together with a transformation engine that transforms STEX documents to the OMDoc format. STEX extends the familiar and time-tried LATEX workflow until the last step of Internet publication of the material: documents can be authored and maintained in STEX using a simple text editor, a process most technical authors are well familiar with. Only the last (publishing) step (which is fully automatic) transforms the document into the unfamiliar XML world. Thus, STEX can serve as a conceptual interface between the document author and OMDoc-based systems: Technically, STEX documents are transformed into OMDoc, but conceptually, the ability to semantically annotate the source document is sufficient.

Chapter 39

Recap of the TEX/LATEX System

TEX [**Knuth:ttb84**] is a document presentation format that combines complex page-description primitives with a powerful macro-expansion facility, which is utilized in LATEX (essentially a set of TEX macro packages, see [**Lamport:ladps94**]) to achieve more content-oriented markup that can be adapted to particular tastes via specialized document styles. It is safe to say that LATEX largely restricts content markup to the document structure[1], and graphics, leaving the user with the presentational TEX primitives for mathematical formulae. Therefore, even though LATEX goes a great step into the direction of a content/context markup format, it lacks infrastructure for marking up the functional structure of formulae and mathematical statements, and their dependence on and contribution to the mathematical context.

But the adaptable syntax of TEX/LATEX and their tightly integrated programming features have distinct advantages on the authoring side:

- The TEX/LATEX syntax is much more compact than OMDoc, and if needed, the community develops LATEX packages that supply new functionality with a succinct and intuitive syntax.
- The user can define ad-hoc abbreviations and bind them to new control sequences to structure the source code.
- The TEX/LATEX community has a vast collection of language extensions and best practice examples for every conceivable publication purpose. Additionally, there is an established and very active developer community that maintains these.
- A host of software systems are centered around the TEX/LATEX language that make authoring content easier: many editors have special modes for LATEX, there are spelling/style/grammar checkers, transformers to other markup formats, etc.

In other words, the technical community is heavily invested in the whole workflow, and technical know-how about the format permeates the community. Since all of this would need to be re-established for an OMDoc-based workflow, the community is slow to take up OMDoc over TEX/LATEX, even in light of the advantages detailed in this book.

[1]supplying macros e.g. for sections, paragraphs, theorems, definitions, etc.

Chapter 40

A LaTeX-based Workflow for XML-based Mathematical Documents

An elegant way of sidestepping most of the problems inherent in transitioning from a LaTeX-based to an XML-based workflow is to combine both and take advantage of the respective values.

The key ingredient in this approach is a system that can transform TeX/LaTeX documents to their corresponding XML-based counterparts. That way, XML-documents can be authored and prototyped in the LaTeX workflow, and transformed to XML for publication and added-value services.

There are various attempts to solve the TeX/LaTeX to XML transformation problem; the most mature is probably Bruce Miller's LaTeXML system [**Miller:latexml**]. It consists of two parts: a re-implementation of the TeX analyzer with all of its intricacies, and an extensible XML emitter (the component that assembles the output of the parser). Since LaTeX style files are (ultimately) programmed in TeX, the TeX analyzer can handle all TeX extensions[1], including all of LaTeX. Thus the LaTeXML parser can handle all of TeX/LaTeX, if the emitter is extensible, which is guaranteed by the LaTeXML binding language: To transform a TeX/LaTeX document to a given XML format, all TeX extensions must have "LaTeXML bindings", i.e. directives to the LaTeXML emitter that specify the target representation in XML.

The sTeX system that we present here supplies a set of TeX/LaTeX packages and the respective LaTeXML bindings that allow to add enough structural information in the TeX/LaTeX sources, so that the LaTeXML system can transform them into documents in OMDoc format.

[=content-markup-LaTeX]

[1]i.e. all macros, environments, and syntax extensions used int the source document

Chapter 41

Content Markup of Mathematical Formulae in TEX/LATEX

The main problem here is that run-of-the-mill TEX/LATEX only specifies the presentation (i.e. what formulae look like) and not their content (their functional structure). Unfortunately, there are no universal methods (yet) to infer the latter from the former. Consider for instance the following "standard notations"[1] for binomial coefficients: $\binom{n}{k}$, ${}_nC^k$, $\mathcal{C}^n_k$, and $\mathcal{C}^k_n$ all mean the same thing: $\frac{n!}{k!(n-k)!}$. This shows that we cannot hope to reliably recover the functional structure (in our case the fact that the expression is constructed by applying the binomial function to the arguments n and k) from the presentation alone short of understanding the underlying mathematics.

The apparent solution to this problem is to dump the extra work on the author (after all she knows what she is talking about) and give her the chance to specify the intended structure. The markup infrastructure supplied by the STEX collection lets the author do this without changing the visual appearance, so that the LATEX workflow is not disrupted. We speak of **semantic preloading** for this process. For instance, we can now write

`\CSum{k}1\infty{\Cexp{x}k}` instead of `\sum_{k=1}^\infty x^k` (41.1)

for the mathematical expression $\sum_{k=1}^\infty x^k$. In the first form, we specify that we are applying a function (`CSumLimits` $\hat{=}$ sum with limits) to four arguments: (*i*) the bound variable k (*ii*) the number 1 (*iii*) ∞ (*iv*) `\Cexp{x}k` (i.e. x to the power k). In the second form, we merely specify hat LATEX should draw a capital sigma character (Σ) whose subscript is the equation $k = 1$ and whose superscript is ∞. Then it should place next to it an x with an upper index k.

Of course human readers (who understand the math) can infer the content structure from the expression $\sum_{k=1}^\infty x^k$ of the right-hand representation in (**??**), but a computer program (who does not understand the math or know the context in which it was encountered) cannot. However, a converter like LATEXML can infer this from the left-hand LATEX structure with the help of the curly braces that indicate the argument structure. This technique is nothing new in the TEX/LATEX world, we use the term "**semantic macro** " for a macro whose expansion stands for a mathematical object. The STEX collection provides semantic macros for all Content-MATHML elements together with LATEXML bindings that allow to convert STEX formulae into MATHML.

[1]The first one is standard e.g. in Germany and the US, the third one in France, and the last one in Russia

Chapter 42

Theories and Inheritance of Semantic Macros

Semantic macros are traditionally used to make TEX/LATEX code more portable. However, the TEX/LATEX scoping model (macro definitions are scoped either in the local group or to the end of the document), does not mirror mathematical practice, where notations are scoped by mathematical environments like statements, theories, or such (see [**Kohlhase:smtl05**] for a discussion and examples). Therefore the sTEX collection provides an infrastructure to define, scope, and inherit semantic macros.

In a nutshell, the sTEX `symdef` macro is a variant of the usual `newcommand`, only that it is scoped differently: The visibility of the defined macros is explicitly specified by the `module` environment that corresponds to the OMDoc `theory` element. For this the `module` environment takes the optional `KeyVal` arguments `id` for specifying the theory name and `uses` for the semantic inheritance relation. For instance a `module` that begins with

```
\begin{module}[id=foo,uses={bar,baz}]
```

restricts the scope of the semantic macros defined by the `\symdef` form to the end of this module given by the corresponding `\end{module}`, and to any other `module` environment that has `[uses={...,foo,...}]` in its declaration. In our example the semantic macros from the modules `bar` and `baz` are inherited as well as the ones that are inherited by these modules.

We will use a simple module for natural number arithmetics as an example. It declares a new semantic macro for summation while drawing on the basic operations like + and − from LATEX. `\Sumfromto` allows us to express an expression like $\sum_{i=1}^{n} 2i-1$ as `\Sumfromto{i}1n{2i-1}`. In this example we have also made use of a local semantic symbol for n, which is treated as an arbitrary (but fixed) symbol (compare with the use of `\arbitraryn` below, which is a new — semantically different — symbol).

```
\begin{module}[id=arith]
  \symdef{Sumfromto}[4]{\sum_{#1=#2}^{#3}{#4}}
  \symdef[local]{arbitraryn}{n}
  What is the sum of the first $\arbitraryn$ odd numbers, i.e.
  $\Sumfromto{i}1\arbitraryn{2i-1}?$
\end{module}
```

is formatted by sTEX to

What is the sum of the first n odd numbers, i.e. $\sum_{i=1}^{n} 2i-1$?

Moreover, the semantic macro `Sumfromto` can be used in all `module` environments that import it via its `uses` keyword. Thus sTEX provides sufficient functionality to mark up OMDoc theories with their scoping rules in a very direct and natural manner. The rest of the OMDoc elements can be modeled by LATEX environments and macros in a straightforward manner.

The sTEX macro packages have been validated together with a case study [**Kohlhase:smtl05**], where we semantically preloaded the course materials for a two-semester course “General Computer

Science I&II" at International University Bremen and transform them to the OMDoc, so that they can be used in the ActiveMath system (see Part VI).

Part XIV

Standardizing Context in System Interoperability

In this project the OMDoc format is used as a content language for the protocol-based integration of mathematical software systems, where the systems offer mathematical services by publishing service descriptions and interoperate by exchanging computation requests and results. The mechanics of the communication and domain-independent part of meaning of these messages is given by a standardized 'interlingua' (which will not concern us here), a possible implementation of the transport layer we have seen in . Here we are interested in the mathematical objects contained in the messages

OMDoc can help with the task of making mathematical objects interoperable, as we have seen in series of experiments of connecting the theorem proving systems ΩMEGA [**BenzmuellerEtAl:otama97**], InKa [**HuSe:itng96**], Pvs [**OwRu92**], *λClam* [**RicSmaGre:ppihol98**], TPS [**AnBi:tatps96**], and CoQ [**CoqManual**] to the MBase system by equipping them with an OMDoc interface. As expected, OpenMath and Content-MathML solve the problem of syntactically standardizing the representation of mathematical objects. For a semantic interoperability we also need to capture their context. This is not a problem for Content-MathML, as the context is already standardized in the MathML recommendation. For OpenMath, the context is given by the set of content dictionaries in use for representing the mathematical objects. Nevertheless mathematical software systems — such as computer algebra systems, visualization systems, and automated theorem provers — come with different conceptualizations of the mathematical objects (see [**KohKoh:esmk05**] for a discussion). This has been in principle solved by supplying a flexible and structured theory level in the form of OMDoc content dictionaries that define necessary mathematical concepts (see ?projects@logics.integrating-libraries? for practical considerations). For systems like theorem provers or theory development environments, where the mathematical objects are axioms, definitions, assertions, and proofs there is another problem: that of standardizing the logical language, which we will discuss in ?projects@logics.integrating?.

Chapter 43

Context Interoperability via Theory Morphisms

As an example for the integration of two mathematical software systems we look at the task of integrating the Pvs and ΩMEGA set theory libraries. This is simpler than e.g. integrating the computer algebra systems MAPLE™ and MATHEMATICA®, since all the conceptualizations and assumptions are explicitly given, but gives an intuition for the difficulties involved. We summarize the situation in Figure **??**, where we compare symbol names for set theory concepts in the two systems. The general problem in such an integration of mathematical software systems consists in

PVS	ΩMEGA	PVS	ΩMEGA
set		subset?	subset
member	in		subset2
empty?	empty	strict_subset?	proper-subset
emptyset	emptyset		superset
nonempty?	not-empty	union	union
full?			union2
fullset			union-over-collection
singleton?	singleton	intersection	intersection
singleton			intersection-over-coll.
complement	set-complement	disjoint?	misses
difference	setminus	meets	
symmetric_difference		add	add-one
	exclunion	remove	

Figure 43.1: Set Theories in ΩMEGA and Pvs

their independent growth over time, leading to differing names, definitions, theory boundaries, and possibly conceptualizations. Most of these particulars are artefacts of constraints imposed by the system (e.g. file lengths). In this situation theory interpretations suggest themselves as a means for theory integration: We can use theory interpretations to establish inclusion into a suitably constructed integration theory. In Figure **??** we have executed this for the set theory libraries of the systems Pvs, ΩMEGA, TPS, and IMPS; we provide an 'integration theory' `mbase:sets` — it provides rationally reconstructed versions of all concepts encountered in the system's libraries — and a set of theory inclusions ρ_* that interpret the system concepts in terms of `mbase:sets`. Note that since the ρ_* are monomorphisms, we can factor any existing theory inclusion (e.g. `pvs:sets` to `pvs:funcs` highlighted in Figure **??**) via the integration theory, using the partial inverse ρ_*^{-1} of ρ_*. For an integration of a set of software systems this refactoring process is repeated recursively from terminal- to initial nodes in the `imports` relation.

Note that *technically* we do not need to change the interface language of the mathematical software systems[1], we only rationally reconstruct their meaning in terms of the new integration

[1]This is important if we want to integrate proprietary software systems, where we have no control over the interfaces.

$$\begin{tikzcd}
\texttt{pvs:funcs} & \texttt{omega:functions} & & \texttt{tps:GWFF334.ABBRV} & \texttt{imps:fun} \cdots \\
 & & \boxed{\text{mbase:sets}} \arrow[ull, "\rho_p^{-1}"] \arrow[ul, "\rho_o^{-1}"'] \arrow[ur, "\rho_t^{-1}"] \arrow[urr, "\rho_i^{-1}"] & & \\
\texttt{pvs:sets} \arrow[uu, dotted] \arrow[urr, "\rho_p"] & \texttt{omega:typed-set} \arrow[uu, dotted] \arrow[ur, "\rho_o"'] & & \texttt{tps:GWFF317.ABBRV} \arrow[uu, dotted] \arrow[ul, "\rho_t"] & \texttt{imps:set} \cdots \arrow[uu, dotted] \arrow[ull, "\rho_i"']
\end{tikzcd}$$

Figure 43.2: Theory Translations for System Integration

theory, which can act as a gold standard for the integration. *Socially* the existence of the new standard theory may prompt a migration to the nomenclature and coverage of the integration theory. Note furthermore, that we have only treated the simple case, where the mathematical conceptualizations underlying the software systems are already explicitly given in a library. For many mathematical software systems the underlying conceptualizations and assumptions are only documented in scientific papers, user manuals, or inscribed into the code. For such systems, **interface theories** that make them explicit have to be developed to pursue the integration strategy presented above. Of course, this process needs a lot of manual labor, but leads to true interoperability of mathematical software systems, which can now re-use the work of others.

Finally note that the integration only works as smoothly as in our scenario, if the systems involved make assumptions about mathematical objects that are compatible with each other. In most cases, incompatibilities can be resolved by renaming concepts apart, e.g. one system considers set union to be a binary operation, while the other considers it as n-ary. Here, the integration theory would supply two distinct (though possibly semantically related) concepts; The theory-based integration approach allows to explicitly disambiguate the concepts and thus prevent confusion and translation errors. In very few cases, systems are truly incompatible e.g. if one assumes an axiom which the other rejects. In this case the theory based integration approach breaks down — indeed a meaningful integration seems impossible and unnecessary.

Chapter 44

A Hierarchy of Logical Languages

In the example above we made use of the fact that theorem proving systems are simpler to deal with than other mathematical software systems, since they encode the underlying assumptions explicitly into mathematical libraries. Unfortunately though, this is only partially true the underlying base logics are usually not treated in this way. Fortunately, logical concepts are treated in OMDoc just like ordinary ones: by content markup in Content-MathML or OpenMath, so that there is no fundamental barrier to treating them as the theory contexts above; we only have to come up with interface theories for them. We have done just that when we equipped various logic-based systems with OMDoc interfaces observing that even though the systems are of relatively different origin, their representation languages share many features:

- TPS and Pvs are based on a simply typed λ-calculus and only use type polymorphism in the parsing stage, whereas ΩMEGA and λ*Clam* allow ML-style type polymorphism.
- ΩMEGA, InKa and Pvs share a higher sort concept, where sorts are basically unary predicates that structure the typed universe.
- Pvs and CoQ allow dependent- and record types as basic representational features.

but also differ on many others: for instance InKa, Pvs, and CoQ explicitly support inductive definitions, but by very different mechanisms and on differing levels. CoQ uses a constructive base logic, whereas the other systems are classical. The similarities are not that surprising, all of these systems come from similar theoretical assumptions (most notably the Automath project [**Bruijn80**]), and inherit the basic setup (typed λ-calculus) from it. The differences can be explained by differing intuitions in the system design and in the intended applications.

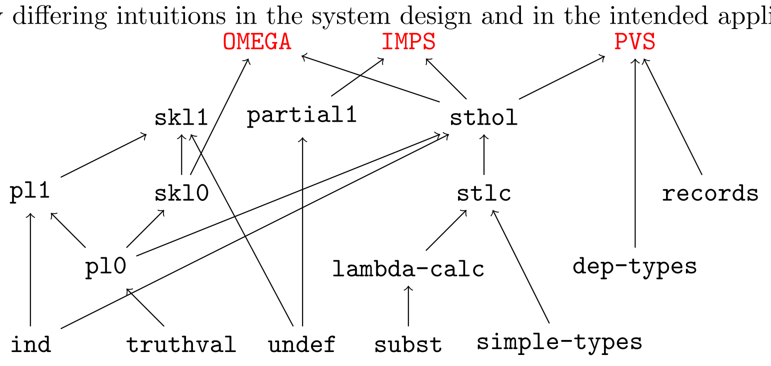


Figure 44.1: A Hierarchy of Logical Languages

We have started to provide a standardized, well-documented set of content dictionaries for logical languages in the OMDoc distribution. These are organized hierarchically, as depicted in

Figure **??**. In essence, the structured theory mechanism in OMDoc is used to create a language hierarchy that inter-relates the various representation formats of existing theorem provers. For instance the simply typed λ-calculus can be factored out (and thus shared) of the representation languages of all theorem proving systems above. This makes the exchange of logical formulae via the OMDoc format very simple, if they happen to be in a suitable common fragment: In this case, the common (OpenMath/OMDoc) syntax is sufficient for communication.

Chapter 45

Logic Interoperability via Logic Morphisms

In theoretical accounts of the integration of logical languages, one finds categorical accounts like the one described in ?projects@hets? or proof-theoretic ones based on definitions like the one below. Both mesh well with the OMDoc representation format and its theory level; we will show this for the proof-theoretic account here.

Definition 45.0.1 A **logical system** $\mathcal{S} = (\mathcal{L},\mathcal{C})$ consists of a language $\mathcal{L}$ (i.e. a set of well-formed formulae) and a calculus $\mathcal{C}$ (i.e. a set of inference rules). A calculus gives us a notion of a $\mathcal{C}$-derivation of $\mathbf{A}$ from $\mathcal{H}$, which we will denote by $\mathcal{D}\colon\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$. Let $\mathcal{S}$ and $\mathcal{S}'$ be logical systems, then a **logic morphism** $\mathcal{F}\colon\mathcal{S} \to \mathcal{S}'$ consists of a **language morphism** $\mathcal{F}^{\mathcal{L}}\colon\mathcal{L} \to \mathcal{L}'$ and a **calculus morphism** $\mathcal{F}^{\mathcal{D}}$ from $\mathcal{C}$-derivations to $\mathcal{C}'$-derivations, such that for any $\mathcal{C}$-derivation $\mathcal{D}\colon\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$ we have $\mathcal{F}^{\mathcal{D}}(\mathcal{D})\colon\mathcal{F}^{\mathcal{L}}(\mathcal{H}) \vdash_{\mathcal{C}'} \mathcal{F}^{\mathcal{L}}(\mathbf{A})$.

The intuition behind this is that logic morphisms transport proofs between logical systems. Logic morphisms come in all shapes and sizes, a well-known one is the relativization morphism from sorted logics to unsorted ones, for instance the morphism $\mathcal{R}$ from sorted first-order logic ($\mathbb{S}FOL$) to unsorted first-order logic (FOL). For every sorted constant $\mathcal{R}$ introduces an axiom e.g. $\mathcal{R}([+\colon\mathbb{N} \to \mathbb{N} \to \mathbb{N}]) = \forall X,Y.\mathbb{N}(X) \wedge \mathbb{N}(Y) \Rightarrow \mathbb{N}(X+Y)$. On formulae sorted quantifications are translated into unsorted ones guarded by sort predicates, e.g. $\mathcal{R}(\forall X_{\mathbb{B}}.\mathbf{A}) = \forall X.\mathbb{B}(X) \Rightarrow \mathcal{R}(\mathbf{A})$. Finally, for proofs we have the correspondence given in Figure **??**, where $\mathbb{A},\mathbb{B},\ldots$ are sort symbols.

$$\mathcal{R}\left(\frac{\mathbf{A}\colon\mathbb{B}\to\mathbb{C} \quad \mathbf{B}\colon\mathbb{B}}{\mathbf{AB}\colon\mathbb{C}}\right) = \frac{\dfrac{\forall X.\mathbb{B}(X) \Rightarrow \mathbb{C}(\mathbf{A}X)}{\mathbb{B}(\mathbf{B}) \Rightarrow \mathbb{C}(\mathbf{AB})} \qquad \mathbb{B}(\mathbf{B})}{\mathbb{C}(\mathbf{AB})}$$

$$\mathcal{R}\left(\frac{\forall X_{\mathbb{B}}.\mathbf{A} \quad \mathbf{B}\colon\mathbb{B}}{[\mathbf{B}/X]\mathbf{A}}\right) = \frac{\dfrac{\forall X.\mathbb{B}(X) \Rightarrow \mathcal{R}(\mathbf{A})}{\mathbb{B}(\mathcal{R}(\mathbf{B})) \Rightarrow \mathcal{R}([\mathbf{B}/X]\mathbf{A})} \quad \mathbb{B}(\mathbf{B})}{\mathcal{R}([\mathbf{B}/X]\mathbf{A})}$$

Figure 45.1: Relativization Morphism on Proofs

In Definition **??** a logical system is a two-partite object consisting of a language and a calculus. In the ontologically promiscuous OMDoc format both parts are represented largely like ordinary mathematically concepts. The notable exception is that proofs have a slightly dual representation, but inference rules of a calculus are still represented as symbols via the Curry-Howard isomorphism (see). Thus a logical system can be represented as an OMDoc theory as we did above, moreover, the logic morphism $\mathcal{R}$ can simply be encoded as a theory inclusion from $\mathbb{S}FOL$ to FOL mapping $\mathbb{S}FOL$ constants for inference rules to FOL terms for proofs. The condition on the form of derivations in Definition **??** now simply takes on the form of a type compatibility condition.

Part XV

Integrating Proof Assistants as Plugins in a Scientific Editor

[13] EdN:13

In contrast to computer algebra systems (CASs), mathematical proof assistance systems have not yet achieved considerable recognition and relevance in mathematical practice. One significant shortcoming of the current systems is that they are not fully integrated or accessible from within standard mathematical text-editors and that therefore a duplication of the representation effort is typically required. For purposes such as tutoring, communication, or publication, the mathematical content is in practice usually encoded using common mathematical representation languages by employing standard mathematical editors (e.g., LaTeX and EMACS). Proof assistants, in contrast, require fully formal representations and they are not yet sufficiently linked with these standard mathematical text editors. Therefore, we have decided to extend the mathematical text editor TeXmacs [**VdH01**] in order to provide direct access from it to the mathematics assistance system ΩMEGA [**OMEGA02**; **SBA-05-a**]. Generally, we aim at an approach that is not dependent on the particular proof assistant system to be integrated [**ABFL-05-a**].

TeXmacs [**VdH01**] is a scientific WYSIWYG text editor that provides professional typesetting and supports authoring with powerful macro definition facilities like in LaTeX. The internal document format of TeXmacs is a SCHEME S-expression composed of TeXmacs specific markup enriched by definable macros. The full access to the document format together with the possibility to define arbitrary SCHEME functions over the S-expressions makes TeXmacs an appropriate text editor for an integration with a mathematical assistance system.

The mathematical proof assistance system ΩMEGA [**OMEGA02**; **SBA-05-a**] provides proof development at a high level of abstraction using knowledge-based proof planning and the proofs developed in ΩMEGA can be verbalized in natural language via the proof explanation system *P.rex* [**Fiedler-01-a**]. As the base calculus of ΩMEGA we use the CORE calculus [**Aut03**; **Aut-05-a**], which supports proof development directly at the *assertion level* [**Hu-96-a**], where proof steps are justified in terms of applications of definitions, lemmas, theorems, or hypotheses (collectively called *assertions*).

Now, consider a teacher, student, engineer, or mathematician who is about to write a new mathematical document in TeXmacs. A first crucial step in our approach is to link this new document to one or more mathematical theories provided in a mathematical knowledge repository. By providing such a link the document is initialized and TeXmacs macros for the relevant mathematical symbols are automatically imported; these macros overload the pure syntactical symbols and link them to formal semantics. In a TeXmacs display mode, where this additional semantic

[13]EDNOTE: project home page: `http://www.ags.uni-sb.de/~omega/projects/verimathdoc`

information is hidden, the user may then proceed in editing mathematical text as usual. The definitions, lemmas, theorems and especially their proofs give rise to extensions of the original theory and the writing of some proof goes along with an interactive proof construction in ΩMEGA. The semantic annotations are used to *automatically* build up a corresponding formal representation in ΩMEGA, thus avoiding a duplicated encoding effort of the mathematical content. Altogether this allows for the development of mathematical documents in professional type-setting quality which in addition can be formally validated by ΩMEGA, hence obtaining *verified mathematical documents*.

Using TEXMACS's macro definition features, we encode theory-specific knowledge such as types, constants, definitions and lemmas in macros. This allows us to translate new textual definitions and lemmas into the formal representation, as well as to translate (partial) textbook proofs into formal (partial) proof plans.

Rather than developing a new user interface for the mathematical assistance system ΩMEGA, we adapt ΩMEGA to serve as a mathematical service provider for TEXMACS. The main difference is that instead of providing a user interface only for the existing interaction means of ΩMEGA, we extend ΩMEGA to support requirements that arise in the preparation of a semi-formal mathematical document. In the following we present some requirements that we identified to guide our developments.

The mathematical document should be prepared directly in interaction with ΩMEGA. This requires that (1) the semantic content of the document is accessible for a formal analysis and (2) the interactions in either direction should be localized and aware of the surrounding context.

To make the document accessible for formal analysis requires the extraction of the semantic content and its encoding in some semi-formal representation suitable for further formal processing. Since current natural language analysis technology cannot yet provide us with the support required for this purpose, we use semantic annotations in the TEXMACS document instead. Since these semantic annotations must be provided by the author, one requirement is to keep the burden of providing the annotations as low as possible.

Due to their formal nature the representations of mathematical objects, for instance, definitions or proofs, in existing mathematical assistance systems are very detailed, whereas mathematicians omit many obvious or easily inferable details in their documents: there is a big gap between common mathematical language and formal, machine-oriented representations. Thus another requirement to interfacing TEXMACS to ΩMEGA is to limit the details that must be provided by the user in the TEXMACS document to an acceptable amount.

In order to allow both the user and the proof assistance system to manipulate the mathematical content of the document we need a common representation format for this pure mathematical content implemented both in TEXMACS and in ΩMEGA. To this end we define a language S, which includes many standard notions known from other specification languages, such as terms, formulas, symbol declarations, definitions, lemmas and theorems. The difference to standard specification languages is that our language S (i) includes a language for proofs, (ii) provides means to indicate the logical context of different parts of a document by fitting the narrative structure of documents rather than imposing a strictly incremental description of theories as used in specification languages, and (iii) accommodates various aspects of underspecification, that is, formal details that the writer purposely omitted. Given the language S, we augment the document format of TEXMACS by the language S. Thus, if we denote the document format of TEXMACS by T, we define a semantic document format $T+S$ as a document format still accepted by TEXMACS.

Ideally this format of the documents and especially the semantic annotations should not be specific to ΩMEGA in order to enable the combination of the TEXMACS extension with other proof assistance systems as well as the development of independent proof checking tools. However, an abstract language for proofs that is suitable for our purposes and that allows for underspecification is not yet completely fixed. So far we support the assertion-level proof construction rules provided by CORE [**Aut03**; **Aut-05-a**]. Thus, instead of defining a fixed language S, we define a language $S(P)$ parametrized over a language P for proofs and define the document format based on $S(C)$, where C denotes the proof language of CORE. This format supports the *static representation* of semantically annotated documents, which can be professionally typeset with TEXMACS.

The TeXmacs document $T + S(C)$ and the pure semantic representation $S(C)$ in the proof assistant must be synchronized. The basic idea here is to synchronize via a diff/patch mechanism tailored to the tree structure of the TeXmacs documents. The differences between two versions ts_i and ts_{i+1} of the document in $T+S(C)$ are compiled into a patch description p of the corresponding document s_i in $S(C)$ for ts_i, such that the application of p to s_i results in s_{i+1} which corresponds to ts_{i+1}. An analogous diff/patch technique is used to propagate changes performed by the proof assistant tool to documents in $S(C)$ towards the TeXmacs document in $T + S(C)$. In order to enable the translation of the patch descriptions, a key-based protocol is used to identify the corresponding parts in $T + S(C)$ and $S(C)$.

Beyond this basic synchronization mechanism, we define a language that allows for the description of specific interactions between TeXmacs and the proof assistant. This language M is a language for structured menus and actions with an evaluation semantics which allows to flexibly compute the necessary parameters for the commands and directives employed in interaction with the proof assistants. The TeXmacs document format $T + S(C)$ is finally extended to $T + S(C) + M$, where the menus can be attached to arbitrary parts of a document and the changes of the documents performed either by the author or by the proof assistants are propagated between $T + S(C) + M$ and $S(C) + M$ via the diff/patch mechanism. Note that this includes also the adaptation of the menus, which is a necessary prerequisite to support context-sensitive menus and actions contained therein.

The goal of the proposed integration is to use Ωmega as a context-sensitive reasoning and verification service accessible from within the first-class mathematical text editor TeXmacs, where the proof assistant adapts to the style an author would like to write his mathematical publication, and to hide any irrelevant system peculiarities from the user. The communication between TeXmacs and Ωmega is realized by an OMDoc-based interface language.

Although so far the proofs are tailored to the rules of the Core system, the representation language in principle is parametrized over a specific language for proofs. We currently replace the Core specific proof languages by some generic notion of proofs, in order to obtain a generic format for formalized mathematical documents. Thereby we started from a language for assertion-level proofs with underspecification [**mkmlanguage**; **AF-05-a**], which we developed from previous experiences with tutorial dialogs about mathematical proofs between a computer and students [**dialog**].

Part XVI

OMDoc as a Data Format for √eriFun

[14] EdN:14

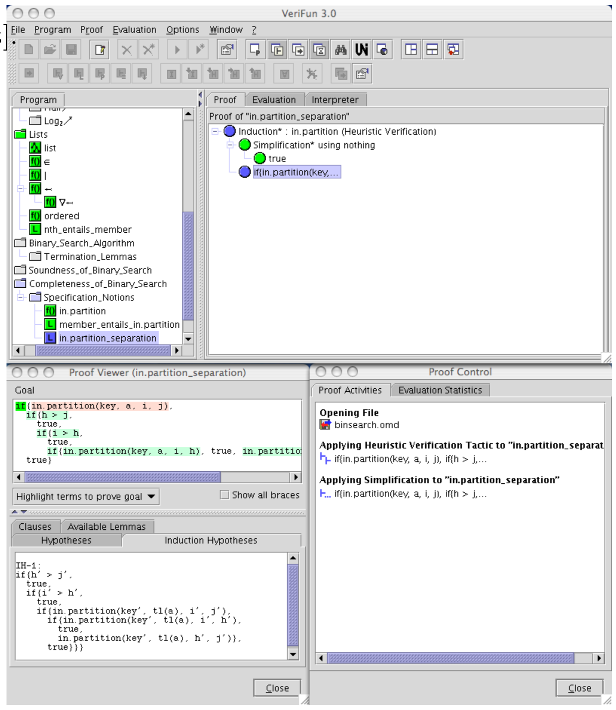
√eriFun (Verification of Functional programs) is a semi-automated system for the verification of programs written in a simple functional programming language $\mathcal{FP}$. The system has been developed since 1998 at the university of Darmstadt for use in education and research. The main design goals are a clearly structured, didactically suited system interface (Figure ??), an easily portable implementation (JAVA) and an easily

[14] EdNote: project page:`http://www.verifun.de/`

but also powerful proof calculus [**WS:VFTut**]. The system's object language consists of a simple definition principle for free data structures, called *sorts* (see), a recursive definition principle for *functions*, and finally a definition principle for statements, called *lemmas*, about the data structures and the functions. To prove a statement `✓eriFun` supports the user with a couple of inference rules aggregated in *tactics*. A collection of *sorts*, *functions*, *lemmas*, and *proofs* is called a `✓eriFun` *program*. Common file commands, which are based on the JAVA binary serialization mechanism, are provided to save and reload intermediate work.

The OMDoc interface for `✓eriFun` described here (see [**NRM:DA05**] for details) was introduced to alleviate the following drawbacks of the former I/O mechanism based on JAVA binary serialization:

- Files are only machine-readable. Thus, e.g. if the files became corrupted by any circumstance, there is no change of a manual repair.
- Files are strongly bound to the version of the system. Thus any internal system modifications make the files unreadable.
- Files are not interchangeable with other theorem provers or other mathematical software systems. Thus the information inside the files are only accessible by `✓eriFun`.


VeriFun 3.0
File Program Proof Evaluation Options Window ?
Program
Log₂↗
Lists
list
∈
|
←
∇←
ordered
nth_entails_member
Binary_Search_Algorithm
Termination_Lemmas
Soundness_of_Binary_Search
Completeness_of_Binary_Search
Specification_Notions
in.partition
member_entails_in.partition
in.partition_separation
Proof Evaluation Interpreter
Proof of "in.partition_separation"
Induction* : in.partition (Heuristic Verification)
Simplification* using nothing
true
if(in.partition(key,…
Proof Viewer (in.partition_separation)
Goal
Highlight terms to prove goal
Show all braces
Clauses Available Lemmas
Hypotheses Induction Hypotheses
Close
Proof Control
Proof Activities Evaluation Statistics
Close

```
if{in.partition(key, a, i, j),
  if{h > j,
    true,
    if{i > h,
      true,
      if{in.partition(key, a, i, h), true, in.partitio
  true}
```

```
IH-1:
if{h' > j',
  true,
  if{i' > h',
    true,
    if{in.partition(key', tl(a), i', j'),
      if{in.partition(key', tl(a), i', h'),
        true,
        in.partition(key', tl(a), h', j')},
      true}}}
```

Opening File
binsearch.omd

Applying Heuristic Verification Tactic to "in.partition_separat
if(in.partition(key, a, i, j), if(h > j,...

Applying Simplification to "in.partition_separation"
if(in.partition(key, a, i, j), if(h > j,...

Figure 45.2: A `✓eriFun` session

`✓eriFun` 's interface to OMDoc can be divided into two parts: Encoding and decoding of `✓eriFun` programs to and from OMDoc respectively.

Chapter 46

Encoding

In a typical session with the system, a user defines a *program* by stipulating the *sorts* and the *functions* of the program, defines *lemmas* about the sorts and the functions of the program, and finally verifies these lemmas and the termination of the functions.

In general a *program* is mapped to two OMDoc files: The first one consists of the user-defined elements[1] and in the second one √eriFun's logic comprising the predefined symbols, the type system and the proof tactics is defined. At each case one √eriFun-generated-OMDoc file is composed of one `theory` element. The name of a user-defined theory can be set by the user, whereas the name of the theory √eriFun is based on is fixed to VAFP.

Functions are declared by `symbol` elements that also introduces the type of the function (). The body of a function is encoded as an OpenMath object inside a `definition` element. The corresponding `symbol` element is referenced by the `definition` element in the `for` and relating termination assertions in the `existence` attribute.

Note that instead of using `name` attributes, which only allow XML simple names, we generate a unique ID. The actual √eriFun names are represented in `presentation` elements or rather their `use` elements (Listing **??**). By using this technique we can use any character string[2] for element names. To cover the whole set of √eriFun fixities (`prefix` (the default), `infix`, `postfix`, `infixl`, `infixr`, and `outfix`) we had to extend the OMDoc format by `infixl` and `infixr`. However, it was not necessary to also add the `outfix` value, but encoding of `outfix` functions is treated slightly different: The name of the function is encoded in the `lbrack` and `rbrack` attribute respectively of the relating `presentation` element and the `use` element is left empty[3].

Lemmata are mapped to `assertion` elements, the value "`lemma`" being assigned to the `type` attribute. The formula of a lemma, analogous to function bodies, is encoded as an OpenMath object inside an `assertion` element.

Particularly convenient is the direct mapping of √eriFun proofs to the OMDoc presentation of proofs. Verifications of lemmas and termination analysis of functions are represented in `proof` elements. The assertion to be proven is referenced in the `for` attribute. VAFP-tactics used inside a proof to achieve the various proof steps (encoded in `derive` elements) are denoted by `method` elements. Parameters heuristically computed by the system or manually annotated by the user are encoded as OpenMath objects and appended to each proof step. Furthermore each proof step in √eriFun is annotated with a sequence of the form $h_1, \ldots h_n, \forall \ldots ih_1, \ldots, \forall \ldots ih_l \vdash goal$ whereas the expressions h_i are the hypotheses, the expressions $\forall \ldots ih_k$ are the induction hypotheses, and the expression *goal* is the goal-term of the sequence. Such a sequent is represented by `assumption` and `conclusion` child-elements respectively of the relating `derive` element.

[1] Actually there are also automatically system-generated elements included, but we may neglect those at this point.

[2] √eriFun has full UNICODE [**Unicode:tuc03**] support

[3] As a consequence the previous mentioned special encoding feature does not hold for outfix functions

Listing 46.1: A polymorphic √eriFun sort

```
structure  list [@value] <=
  ∅,
  [ infixr ,100]  :: (hd : @value, tl  :  list [@value])
```

Sorts are wrapped inside `adt` elements. At this point this integration process provoked two further adaptations of the OMDoc standard. On the one hand, in contrast to OMDoc, sorts in √eriFun could be polymorphic (Listing ??). This led to the additional, optional `parameters` attribute of an `adt` element (Listing ??). Within this new attribute one can declare by a comma separated list the names of type variables of the abstract data type.

Listing 46.2: A polymorphic OMDoc ADT

```
<adt xml:id="vf7b9f3e59−e78e−4221−8064−7fa0c5689f5d.adt" parameters="value">
  <sortdef name="vf7b9f3e59−e78e−4221−8064−7fa0c5689f5d" type="free">
    <constructor name="vf8a6673ac−c1d9−4698−b6ee−90213539a984"/>
    <constructor name="vf38164505−4983−417f−8bdc−6a42b046e933">
      <argument>
        <type system="simpletypes">
          <OMOBJ xmlns="http://www.openmath.org/OpenMath">
            <OMV name="value"/>
          </OMOBJ>
        </type>
        <selector name="vf9fc4c672−207f−45c0−ae61−1f675fde7aed" total="yes"/>
      </argument>
      <argument>
        <type system="simpletypes">
          <OMOBJ xmlns="http://www.openmath.org/OpenMath">
            <OMA>
              <OMS cd="VeriFun" name="vf7b9f3e59−e78e−4221−8064−7fa0c5689f5d"/>
              <OMV name="value"/>
            </OMA>
          </OMOBJ>
        </type>
        <selector name="vf55767f3a−b019−4308−88f9−d68ee0db595e" total="yes"/>
      </argument>
    </constructor>
  </sortdef>
</adt>
```

On the other hand, the child elements of a `constructor` element had to be expanded by an additional `type` element to specify the type of the formal parameter of the parent `constructor` element. Listing ?? illustrates the corresponding `presentation` elements of the ADT in Listing ??.

Listing 46.3: Representation of √eriFun names to OMDoc

```
<presentation for="#vf7b9f3e59−e78e−4221−8064−7fa0c5689f5d" role="applied">
  <use format="VeriFun">list</use>
</presentation>
<presentation for="#vf8a6673ac−c1d9−4698−b6ee−90213539a984" role="applied"
               bracket−style="math" precedence="1" fixity="prefix" lbrack="(" rbrack=")">
  <use format="VeriFun">∅</use>
</presentation>
<presentation for="#vf38164505−4983−417f−8bdc−6a42b046e933" role="applied"
               bracket−style="math" precedence="100" fixity="infixr" lbrack="(" rbrack=")">
  <use format="VeriFun">::</use>
</presentation>
<presentation for="#vf9fc4c672−207f−45c0−ae61−1f675fde7aed" role="applied"
               bracket−style="math" precedence="1" fixity="prefix" lbrack="(" rbrack=")">
  <use format="VeriFun">hd</use>
</presentation>
<presentation for="#vf55767f3a−b019−4308−88f9−d68ee0db595e" role="applied"
               bracket−style="math" precedence="1" fixity="prefix" lbrack="(" rbrack=")">
  <use format="VeriFun">tl</use>
</presentation>
```

Chapter 47

Decoding

The decoding of a √eriFun program represented in OMDoc is reverse to the encoding mechanism. First we create an empty program and then start the sequential decoding of each `adt`, `symbol` and its relating `definition`, and `assertion` element back into the $\mathcal{FP}$ syntax. After a successful reconstruction of an element it is appended to the current program. Right after such an insertion we check for a `proof` element containing a reference to this new program element. If a proof exists, we re-play all the proof steps and associate the recreated √eriFun proof to the corresponding program element.

One aspect of this decoding exercise is worth mentioning here. The √eriFun system also benefited by the development of the OMDoc standard: Revelation of bugs deep in the system! Especially $\mathcal{FP}$ parser errors and inconsistencies in proof tactics applications could be discovered. Maybe those errors would never have been detected, because in most cases the user is not able to produce them manually, but this errors are automatically generated by the system. So with the assistance of the strict encoding and decoding to and from OMDoc respectively we were able to achieve a much more robust verification system.

By the integration of the open content Markup language OMDoc into the semi-automated theorem prover √eriFun, we made the system more reliable and facilitate the participation in the mathematical network to serve as yet another service. Functional programs and especially proof of statements created in √eriFun are now open to the public. The data is human-readable, machine-understandable, no longer subjected to a particular version of the system. Thus, √eriFun generated knowledge became accessible, robust, interchangeable and transparent.

Index

ΩMEGA, 43, 56–58, 60–62, 79

abstract
  data
    type, 22
ACTIVEMATH, 6, 29–35, 49, 55
`adt`
  element, 65
ALGEBRA, 13
analysis
  formal, 61
  natural language, 61
analyzer, 53
annotated
  semantically (), 61
annotation
  semantic, 61
`mq:anyorder`
  attribute, 18, 20
application
  web, 27
`apply`
  element, 18, 19
artificial
  intelligence, 44
assertion, 56
  level, 60
`assertion`
  element, 41, 64, 65
`assumption`
  element, 64
authoring
  semantic (), 49
automated
  theorem
    provers, 56
axiom, 56
`axiom-inclusion`
  element, 41

base
  knowledge, 44
Bibliographic
  Record
    Schema, 11
browsing, 10

calculus, 60
CAS, 25–27
cascades
  finite-state, 12
CASL, 40, 41, 43, 45, 46
CCAPS, 48
change
  management, 21, 23
  management of, 43
clipboard
  system, 11
`CMP`
  element, 6, 35
communication, 25, 60
comorphism, 45
`competency`
  element, 30
complete
  configuration, 22
Complex Theories
  spec Module CTH, 40, 45
Component
  Presentation, 29
  Tutorial, 29
component
  shared, 40
computation
  request, 56
computer algebra
  system, 56
conceptual
  structure, 6
`conclusion`
  element, 64
configuration
  complete, 22
  consistent, 22
  management, 21
CONNEXIONS, 19
consequence
  morphism, 41
consistent
  configuration, 22
construct
  programming, 26
`constructor`
  element, 65
content
  language, 56
  semantic, 61
Content in

PowerPoint, 48
Content MathML, 19, 54, 56, 58
context
  dynamic, 26
  dynamic (), 27
  static (), 27
control
  version, 21
CoQ, 56, 58
Core, 60–62, 77
Core, 60
CPoint, 3, 5, 48–51
CPointAuthor, 50, 51
CPointBasic, 51
CPointGraphs, 50
CPointImport, 50
CPointNotes, 50
CPointStudent, 51
CTH (Complex Theories)
  spec Module, 40, 45
Curry-Howard
  isomorphism, 59

data
  abstract (), 22
definition, 56
`definition`
  element, 29, 64, 65
`derive`
  element, 64
description
  logic, 44
  service, 56
development
  graph, 40, 41, 43
  time, 40
Development Graphs
  spec Module DG, 45
  sped Module DG, 40
DG (Development Graphs)
  spec Module, 45
  sped Module, 40
Dgrl, 41, 43
`difficulty`
  element, 30
Digital Media in Education, 48
DocBook, 26
document
  library, 11
  manager, 27
  semantic (), 61
  server, 26
  structure, 22
  structured, 23
  view, 11
dynamic
  context, 26
    manager, 27
  mathematical
    context, 25

editing, 10
Emacs, 60
emitter, 53
entailment
  system, 45
evaluation
  semantics, 62
evolutionary
  process, 41
`example`
  element, 31
`existence`
  attribute
    on `definition`, 64
external
  mathematical
    service, 26

`field`
  element, 30
finite-state
  cascades, 12
FireFox, 10, 13, 32
`FMP`
  element, 6
`for`
  attribute
    on `definition`, 64
    on `proof`, 64
formal
  analysis, 61
  representation, 61
  semantics, 60
  software
    development, 40
formalization, 45

`mq:generic`
  attribute, 18, 19
global
  theorem
    link, 41
goal
  learning, 29
GoTo, 49
graph
  development, 40, 41, 43

HetCasl, 45
Hets, 23, 43–47
higher-order
  logic, 41
HTML, 15, 32, 34, 35

icmap, 32
IkeWiki, 36, 37
implicit

knowledge, 48
imports
  element, 41, 57
IMPS, 57
in-the-large
  structuring, 22
infix
  attribute value
    for fixity on presentation, 64
infixl
  attribute value
    for fixity on presentation, 64
infixr
  attribute value
    for fixity on presentation, 64
inheritance
  semantic (), 54
INKA, 41, 56, 58
institution, 45
integration
  protocol-based, 56
  theory, 57
  type, 42
intelligence
  artificial, 44
interactive
  mathematical
    document, 25
interactivity, 25
interface, 5
interlingua, 56
isomorphism
  Curry-Howard, 59

JAVA, 63
JEDIT, 34
JEDITOQMATH, 34, 35

Knowledge
  Mathematical (), 4
knowledge
  base, 44
  implicit, 48
  mathematical (), 60

λClam, 56, 58
language
  content, 56
LATEXML, 53, 54
lbrack
  attribute
    on presentation, 64
Learner
  Model, 29
learning, 29
  goal, 29
lemma
  attribute value
    for type on assertion, 64
level
  assertion, 60
library
  document, 11
logic
  description, 44
  higher-order, 41
  morphism, 59
  translation, 45

macro
  semantic, 54
management
  change, 21, 23
  configuration, 21
management of
  change, 43
manager
  document, 27
MAPLE™, 12, 13, 56
markup
  semantic, 48
math
  element, 19
math-enabled
  web
    browser, 26
MATHDOX, 6, 25–27
Mathematical
  Knowledge
    Management, 4
mathematical
  dynamic (), 25
  external (), 26
  interactive (), 25
  knowledge
    repository, 60
  practice, 60
  publication, 62
  service, 27, 56
  symbol, 60
  text, 60
  text-editors, 60
  verified (), 61
MATHEMATICA®, 12, 13, 25, 56
MATHML
  content, 19, 54, 56, 58
MATHML, 6, 7, 11, 12, 17–20, 32, 54, 56
MATHWEB, 15
MAXIMA, 12, 13, 31
MAYA, 5, 40–43, 45
MBASE, 5, 6, 15, 16, 41, 42, 56
metadata
  element, 48
method
  element, 64
misconception
  element, 30
MMISS, 6, 21, 22, 24

Model
  Learner, 29
morphism
  consequence, 41
  logic, 59
mOZart, 15
Mozilla, 5, 10, 12, 13, 32
`mq:anyorder`
  attribute, 18, 20
`mq:generic`
  attribute, 18, 19

`name`
  attribute
    on `symbol`, 64
narrative
  structure, 6
natural language
  analysis, 61
navigation, 50

obligation
  proof, 40, 41
OMDoc, 4–8, 10, 11, 13, 15, 24, 26, 29–42, 45, 46, 48–50, 52–56, 58, 59, 62–66
`omdoc`
  element, 37
Ωmega, 60
`omgroup`
  element, 48
`omtext`
  element, 29
ontologically
  promiscuous, 59
ontology, 21
OpenMath, 7, 12, 13, 17, 18, 25–27, 31, 32, 34, 56, 58, 64, 73
OpenOffice, 13
OQMath, 7, 34, 35
`outfix`
  attribute value
    for `fixity` on `presentation`, 64
Owl, 38
Oz, 15

`swim:page`
  element, 37
`parameters`
  attribute
    on `adt`, 65
PDF, 32
pickling, 15
`postfix`
  attribute value
    for `fixity` on `presentation`, 64
PowerPoint
  Content in , 48
MS PowerPoint, 48–51
practice
  mathematical, 60
`prefix`
  attribute value
    for `fixity` on `presentation`, 64
preloading
  semantic, 53
Presentation
  Component, 29
presentation, 48
`presentation`
  element, 64, 65
P.Rex, 60
process
  evolutionary, 41
programming
  construct, 26
promiscuous
  ontologically, 59
proof, 56
  obligation, 40, 41
`proof`
  element, 64, 65
proof assistance
  system, 60
proof-theoretic, 59
protocol-based
  integration, 56
publication, 60
  mathematical, 62
Pvs, 56–58

QMath, 5, 7–10, 12, 13, 34, 39

`rbrack`
  attribute
    on `presentation`, 64
RDF, 11, 36–38
Record
  Bibliographic (), 11
representation
  formal, 61
  static, 61
request
  computation, 56
requirement
  specification, 22, 40

Scheme, 60
searching, 10
Semantic
  Web, 44
semantic
  annotation, 61
  authoring
    tool, 49
  content, 61
  document
    format, 61
  inheritance

relation, 54
macro, 54
markup, 48
preloading, 53
SEMANTIC MEDIAWIKI, 37
semantically
annotated
documents, 61
semantics
evaluation, 62
formal, 60
SENTIDO, 5, 9–14
server
document, 26
service
description, 56
mathematical, 27, 56
set
theory, 56
shared
component, 40
SIETTE, 30
software
formal (), 40
spec Module
CTH (Complex Theories), 40, 45
DG (Development Graphs), 45
specification
requirement, 22, 40
system, 40
sped Module
DG (Development Graphs), 40
state, 26
static
context
manager, 27
representation, 61
structure
conceptual, 6
document, 22
narrative, 6
structured
document, 23
structuring
in-the-large, 22
SVG, 32
SWIM, 36–39
`swim:page`
element, 37
symbol
mathematical, 60
`symbol`
element, 64, 65
system
clipboard, 11
computer algebra, 56
entailment, 45
proof assistance, 60
specification, 40
visualization, 56

TEXMACS, 60
TEXPOINT, 50
text
mathematical, 60
text-editors
mathematical, 60
theorem
automated (), 56
global (), 41
THEOREMA, 13
theory, 41
integration, 57
set, 56
`theory`
element, 37, 54, 64
`theory-inclusion`
element, 41
time
development, 40
toolbar, 49
TPS, 56–58
translation
logic, 45
Tutorial
Component, 29
tutoring, 60
type
integration, 42
`type`
element, 65
`type`
attribute
on `assertion`, 64

UNICODE, 7, 13, 64
`use`
element, 64

variant, 23
verified
mathematical
document, 61
version
control, 21
view
document, 11
visualization
system, 56
VSE-SL, 40, 41

Web
Semantic, 44
web
application, 27
math-enabled (), 26
WIRIS, 31

workflow, 53
WYSIWYG, 60

XHTML, 6, 11, 32
XML, 8, 10, 15, 18, 19, 25, 26, 34, 35, 52, 53, 64
XML-RPC, 29
XPath, 11
XSLT, 11, 33

Yacas, 12, 13, 31