INFORMATION SOCIETY TECHNOLOGIES
(IST)
PROGRAMME

Project IST-2001-33562 MoWGLI

Preliminary Report on Application Scenarios and Requirement Analysis

Main Authors:
A. Asperti, Y. Bertot, H. Geuvers, E. Gimenez, H. Herbelin, P. Libbrecht,
L. Padovani, C. Sacerdoti Coen, I. Schena, B. Schutz, C. Weyher.

Project Acronym: MoWGLI
Project full title: Mathematics On the Web: Get it by Logic and Interfaces
Proposal/Contract no.: IST-2001-33562 MoWGLI

Contents

1 Introduction

The World Wide Web and its technology is rapidly changing the way mathematicians deal with knowledge. Sophisticated portals, search engines, and databases currently provide an easy and friendly way for finding information in the fast-growing realm of mathematical information (A.M.Odlyzko [21] estimated that the amount of published mathematics doubles every ten-fifteen years). The distinguished experimental mathematician P.Borwein has recently declared [4]: “I stopped reading mathematics journals some years ago . . . MathSciNet replaced browsing”. MathSciNet, developed under the auspices of the American Mathematical Society, is a typical example of a state-of-the-art searchable web database: it covers 1799 journals, offering links to 192200 original articles, and indexing over 300000 authors (over 20000 new items have been added since the beginning of 2002). However, MathSciNet has clear limitations: on one side the HTML interface does not adequately support mathematical notation, with the consequence that abstracts and reviews are often unpleasant to read and sometimes hard to understand; on the other side, it does not offer any support for searching by mathematical *content*, instead of mathematical (key-)words. The first problem is likely to be rapidly solved by emerging display technologies like MathML. The second problem is far more complicated, concerning the very way mathematics is actually encoded: currently, almost all mathematical documents available on the Web are marked up only for presentation, severely crippling the potentialities for automation, interoperability, sophisticated searching mechanisms, intelligent applications, transformation and processing. The goal of MoWGLI is exactly to explore the possibilities for overcoming these limitations, passing from a machine-readable to a machine-understandable representation of the information, and developing the technological infrastructure for its exploitation.

The problem of suitably *representing* mathematical knowledge is an essential prerequisite for any kind of *mathematical knowledge management*, a new topic of research which is recently attracting an increasing interest among the mathematical and computer science community. The success of the first International Conference on Mathematical Knowledge Management, currently at its second edition (`http://www.cs.unibo.it/MKM03`), and the recent creation of the European Network IST-2001-37057 KMK-NET are clear hints of this trend. But the real novelty is the, comprehensibly timid, but equally evident interest that maintainers of digital libraries are paying for a more contenutistic description of mathematics.

On a completely different level, there is a compelling need of integration between the current tools for automation of formal reasoning and mechanisation of mathematics (proof assistants and logical frameworks) and the most recent technologies for the development of web applications and electronic publishing. Currently, libraries in logical frameworks are usually saved in two formats: a textual one, in the specific tactical language of the proof assistant, and a compiled (proof checked) one in some internal, concrete representation language. Both representations are obviously unsatisfactory, since they are too oriented to the specific application: the information is not directly available, if not by means of the functionalities offered by the system itself. This is in clear contrast with the main guidelines of the modern Information Society, and its new emphasis on *content*.

The requirements, are in this case somewhat dual to the case of digital libraries: we have here a strongly formalised and machine-understandable encoding, that however is too application dependent, prevents any kind of interoperability, and any kind of modularisation even of those functionalities (like e.g. displaying or searching) which are largely independent from the specific formal system.

One of the main aim of MoWGLI is to look for a possible common ground, provided, in our view, by a semi-formal *content* description of mathematical information, between the distant communities of automation of formal reasoning and digital libraries management. Of course, it is important to be aware that arriving to a real integration between these two worlds is a long term goal which is far off the limited objectives of MoWGLI. In a recent paper [23] of the Bulletin of Symbolic Logic devoted to the prospects of proof theory and logic for the new century, the following prediction was made:

> "Computer databases of mathematical knowledge will contain, organize and retrieve most of the known mathematical literature by 2030"

MoWGLI is just meant to be a very preliminary step in this direction (and the fact of having put the two communities together inside a single project is already a success). What we may reasonably expect is just to elucidate the problems and the requirements for a "content" description of mathematics, hopefully reaching some agreement on a standard encoding for some fragments, and developing tools which could allow to test the new potentialities offered by this representation.

1.1 Semantics, Content and Presentation

By the previous discussion, it should be clear that we are dealing with at least three different encoding of mathematical information: semantics, content and presentation.

The distinction between content and presentation is clear, and it was already evinced in the Mathematical Markup Language (MathML) recommended by the World Wide Web as the intended standard for encoding of mathematical expressions on the Web. The same document, however, is quite confused in addressing the intended *semantics* of the mathematical operators.

By semantics, we mean here a really formal encoding, such as it could be profitably used by a machine in order to understand and manipulate the mathematical content of the information (in particular, it is not just a pedantic foundational issue, but a really practical automation problem). If we consider a typical content element of MathML such as, e.g. the logical implication, it is clear that the language is not trying to capture a *formal* semantics for it (no reference to, say, classical first order logic is explicitly made in the Recommendation), but merely the *structure*, i.e. the abstract syntax, of an implicative formula. Even "simpler" mathematical operators like a sum, may be ambiguously interpreted if there is no agreement on the underlying data structure (e.g. integers in arbitrary or fixed precision). The MathML specification is not entering in these details, clearly leaving to the application the burden to give a formal interpretation to the operators, hopefully "reasonably" close to the "intended", informal meaning.

In particular, the distinction between content and semantics makes sense only in view of the intended elaboration of the information. From the point of view of a digital library, the main reason for adding "content" is to facilitate some kind of searching and retrieving functionalities which could not be otherwise implemented disposing of a mere presentational markup. A second, mid-term reason, could be the possibility of passing some expressions into a computer algebra system, to check their formal validity. A really long-term goal (definitely beyond the scope of MoWGLI) is to be able to formally check proofs. For the first kind of elaboration, no strictly formal semantics is required: a "content" description capturing the intended name and structural syntax of the mathematical operator is enough. Pushing too far a semantical distinction between "similar" operators, or refining too much the descriptive markup, could even have a negative impact on the effectiveness of the research. For the purpose of giving an expression as input to a computer algebra system, some kind of formal semantics is required, but in this case we could assume that the intended domain of the application is sufficiently clear to allow a precise mapping from content to semantics. The real problems, and the necessity of making a clear distinction between content and semantics only arise when we enter in the delicate problem of checking proofs, that requires to give a precise meaning even to common datatypes or operators, which are hardly "primitive" in the logical systems. In this case, the semantical markup may strictly depend both on the particular logical, foundational system and by the precise formalisation of the mathematical notion in the given logical framework.

Since the goal of attaining a real integration between digital libraries and tools for automatic checking of proofs is clearly beyond the scope of MoWGLI, the real question is the following:

> Does a content description fruitful for the intended usage inside digital libraries have any interesting meaning and usage from the point of view of tools for the automation of formal reasoning?

If the answer is negative, there is no common ground for a fruitful collaboration for the two communities yet, and we shall have to wait for more substantial progresses of the latter tools.

However, the claim of MoWGLI is that there are good reasons for answering positively to the previous question.

First of all, a semi-formal content level appears to be a main component of the typical mathematical practice. Typically, if we meet an integration symbol in a mathematical statement we hardly wonder what kind of integration has been dealt with by the author. It is only at a second stage, if and when we possibly meet some technical and obscure formal detail, that we may come back to check the precise intended meaning of the symbol. So, a pseudo-formal contenutistic level seems indeed to belong to the usual practice of mathematics. We could even claim that these kind of "notational abuse" is in some case a useful and fruitful tool of mathematics.
In general, given any operator of the mathematical language, it looks possible to associate to the operator a content (e.g. "some" kind of integration) and a semantics ("what" kind of integration). It is indeed the fact that different integrations have a strong and common nature that justifies the fact of calling - all of them - integrations.

Secondly, this "content" layer provides an essential abstraction over the strictly formal description, providing a simple way for improving the modularity of the applications for several important elaboration issues, such as rendering and searching/retrieving operations. In particular, after defining a mapping from "semantics" to "content", all the tools developed for the latter layer may be directly and profitably applied to the former.

Least, but not last, the "content" layer fixes names and structure, that is one of the most cumbersome aspects of mathematical standardisation.

1.2 Beyond expressions

Another important topic is the necessity to go beyond the mere description of mathematical *expressions*. Mathematics is a richly structured language, not only at the level of expressions, but at several different layers, comprising proof, statements, theories, and so on. It looks extremely important to be able to introduce a reasonable markup for all these layers, especially from the point of view of automation of formal reasoning. A precise discussion of these layers will be the subject of the report on "Structure and Metastructure of Mathematical Developments".

It is however important to point out that, even at this level, there is probably a reasonable space for a "content" description as opposed to a "semantic" one.

Current standardisation efforts such as the "Weak Type Theory" of R. Nederpelt and F. Kamareddine [20] are exactly aimed to provide a sort of intermediate language between the natural language of the mathematician and the formal language of the logician.

The OMDoc format [3, 14] (see section 4.3) adds content markup at the level of mathematical statements and theories. It uses OpenMath and content MathML for formula representation. Whereas weak type theory attempts a linguistically motivated treatment of classification of everyday mathematical language constructions, OMDoc adds XML-based representation primitives and a basic infrastructure for document relations.

2 The role of XML technology

Before entering in a more detailed description of a few typical application scenarios, it is worth to spend a few words on the role of XML technology in the project. As a matter of fact, XML is the main new technological ingredient which has allowed to start anew the ambitious goal of mathematical knowledge management on a *solid* technological ground, opening new perspectives which were just unrealistic before.

XML [36], whose development started in 1996 by a Working Group under the aegis of the World Wide Web Consortium, was conceived as a streamlined version of SGML[1] designed to simplify transmission and manipulation of structured documents over the Web. Its purpose is to encode information according to their structure and content, via markup tags and hyperlinks. The markup makes possible manipulation, analysis, storage, linking, and transformation of all media, obviously comprising text.

Once in XML, the information is directly available to everybody (you have no longer to rely on the facilities offered by the specific applications). You may already start to develop your own applications on top of it (like complex spiders looking for specific structures inside the terms of the library, or tools for extracting metadata, or interfaces with databases, or whatever). All these applications are likely to be re-usable or easily adaptable to whatever logical dialect has been used to encode the information.

In particular, by choosing XML as a central technology for storing, retrieving and processing mathematical documents, we expect major benefits in all of the following crucial areas:

Interoperability If having a common representation layer is not the ultimate solution to all interoperability problems between different applications, it is however a first and essential step in this direction.

Standardisation Having a common, application independent encoding for mathematical proofs, similar software tools could be applied to different logical dialects, regardless of their concrete nature. This would be especially relevant for those operations like searching, retrieving, displaying or authoring (just to mention a few) that are largely independent of the specific logical system.

Publishing There exist sophisticated Web-publishing technologies based on XML, such as Stylesheets, MathML, Formatting Objects. Among the benefits of these technologies, the most important ones for HELM are media independence and notational and stylistic customisation. The combination of client-side customisable rendering languages with the flexibility of user-provided stylesheets can be profitably used to solve, in a *standard*, *extensible* way the annoying notational problems that traditionally afflict formal mathematics.

Searching & Retrieving The World Wide Web Consortium is currently doing a big effort in the Metadata and Semantic Web area. Languages such as the Resource Description Framework or XML-Query are likely to produce innovative technological solutions in this field.

Modularity The "XML-ization" process should naturally lead to a substantial simplification and re-organisation of the current, "monolithic" architecture of logical frameworks. The

[1]The Standard Generalized Markup Language (SGML) is an international Standard which has been widely used in high-end areas of information management and publishing.

many different and often loosely connected functionalities of these complex programs (proof checking, proof editing, proof displaying, search and consulting, program extraction, and so on) could be clearly split in more or less autonomous tasks, possibly (and hopefully!) developed by different teams, in totally different languages. This is the new *content-based* architecture of future systems.

2.1 XML and the standardisation effort

XML is a meta-language. The syntactical structure of domain specific languages may be defined by the so called Document Type Definitions [36], i.e. grammar specifications describing the structure and attributes of elements for each specific instance of XML.

It is important to understand that the same information may be encoded in different ways, according to its intended use. The point of XML is that of exploiting the potentialities of all these different encodings, providing tools (such as stylesheets) for passing from one encoding to the other. So, standardisation must be pushed at specific descriptive layers, for specific uses, carefully avoiding the hopeless quest for a "universal" mathematical language (the standardisation we are pursuing is first of all technological, and not logical).

As an example, let us consider the simple mathematical expression $\forall a(a * 0 = 0)$.

The typical content encodings of this expression into MathML and OpenMath – instances of XML explicitly conceived for the management of mathematical expressions on the Web (see sections 4.1 and 4.2) – are the following:

```
<math>                             <OMOBJ>
 <apply>                            <OMBIND>
  <forall/>                          <OMS cd="quant1" name="forall"/>
  <bvar>                             <OMBVAR>
   <ci type="integer">a</ci>          <OMV name="a"/>
  </bvar>                            </OMBVAR>
  <apply>                            <OMA>
   <eq/>                              <OMS cd="relation1" name="eq"/>
   <apply>                            <OMA>
    <times/>                           <OMS cd="arith1" name="times"/>
    <ci type="integer">a</ci>          <OMV name="a"/>
    <cn>0</cn>                         <OMI>0</OMI>
   </apply>                           </OMA>
   <cn>0</cn>                         <OMI>0</OMI>
  </apply>                           </OMA>
 </apply>                           </OMBIND>
</math>                            </OMOBJ>
```

The same expression inside a typical Proof Assistant may have a sensibly different representation. For example, in the case of Coq:

1. we are dealing with a strongly typed foundational system, where types must be explicited

2. types in Coq can be instances of a primitive construction called "mutual inductive type". This is the case for natural numbers, and also for particular relations such as equality.

3. the constant “0” is just the first constructor of (the first and in this case unique mutual inductive) type of natural numbers.

4. multiplication is not a primitive operation

5. bound variables are internally represented by means of DeBruijn indexes.

So, a more faithful encoding of the same information may look as follows:

```
<Definition name="mult_n_O" id="i0" params="">
 <type>
  <PROD id="i32">
   <source>
    <MUTIND uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0"/>
   </source>
   <target binder="a">
    <APPLY id="i34">
     <MUTIND uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0"/>
     <MUTIND uri="cic:/Coq/Init/Logic/Equality/eq.ind" noType="0"/>
     <APPLY id="i37">
      <CONST uri="cic:/Coq/Init/Peano/mult.con"/>
      <REL value="1" binder="a" id="i39"/>
      <MUTCONSTRUCT uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0" noConstr="1"/>
     </APPLY>
     <MUTCONSTRUCT uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0" noConstr="1"/>
    </APPLY>
   </target>
  </PROD>
 </type>
</Definition>
```

No one of the previous XML encodings could be claimed to be better than the other, they are at the extremes of content/semantic markup of mathematical expressions and encode different information, for different intended uses. In particular, the MathML encoding provides a typical example of “content” description, as opposed to the strictly formal, “semantical” encoding of the second dialect. It is however fairly easy to map semantics over content, thus recovering the intended “content” of the formal expression and the possibility to uniformly use “content” tools for e.g. searching or publishing issues.

Moreover, while – at the semantical level – we cannot escape, both theoretically and practically, the multi-lingual environment of the foundations of mathematics, we may reasonably hope to reach a higher degree of linguistic standardisation at “content” level (provided we renounce to attribute a strong semantical meaning to this layer).

3 Application Scenarios

There are three types of 'mathematical-content' documents that we will consider in MoWGLI: real scientific papers from different fields such as mathematics and physics; formalised mathematics (within the Coq proof-assistant); documentation for security evaluation of IT products. For each one we present a small sample document with a listing of the desired features that we should provide and the corresponding content requirements.

3.1 Sample document 1

This sample document has been extracted from:

> Luc Blanchet, "Gravitational Radiation from Post-Newtonian Sources and Inspiralling Compact Binaries", *Living Rev. Relativity* **5**, (2002), 3. [Online Journal Article]: http://www.livingreviews.org/Articles/Volume5/2002-3blanchet/index.html
> Section 4.1.: The post-Minkowskian solution

We insert the ansatz (29) into the vacuum Einstein field equations (12, 13) *i.e.* with $\tau^{\alpha\beta} = c^4/(16\pi G)\Lambda^{\alpha\beta}$, and we equate term by term the factors of the successive powers of our book-keeping parameter G. We get an infinite set of equations for each of the $h_n^{\alpha\beta}$'s: $\forall n \geq 2$,

$$\Box h_n^{\alpha\beta} = \Lambda_n^{\alpha\beta}[h_1, h_2, \ldots, h_{n-1}], \tag{30}$$

$$\partial_\mu h_n^{\alpha\mu} = 0. \tag{31}$$

The right-hand side of the wave equation (30) is obtained from inserting the previous iterations, up to the order $n-1$, into the gravitational source term. In more details, the series of equations (30) reads

$$\Box h_2^{\alpha\beta} = N^{\alpha\beta}[h_1, h_1], \tag{32}$$

$$\Box h_3^{\alpha\beta} = M^{\alpha\beta}[h_1, h_1, h_1] + N^{\alpha\beta}[h_1, h_2] + N^{\alpha\beta}[h_2, h_1], \tag{33}$$

$$\begin{aligned} \Box h_4^{\alpha\beta} &= L^{\alpha\beta}[h_1, h_1, h_1, h_1] \\ &\quad + M^{\alpha\beta}[h_1, h_1, h_2] + M^{\alpha\beta}[h_1, h_2, h_1] + M^{\alpha\beta}[h_2, h_1, h_1] \\ &\quad + N^{\alpha\beta}[h_2, h_2] + N^{\alpha\beta}[h_1, h_3] + N^{\alpha\beta}[h_3, h_1] \\ &\quad \vdots \end{aligned} \tag{34}$$

The quadratic, cubic and quartic pieces of $\Lambda^{\alpha\beta}$ are defined by Eq. (16).

Let us now proceed by induction. Some n being given, we assume that we succeeded in constructing, from the linearized coefficient h_1, the sequence of post-Minkowskian coefficients h_2, h_3, …, h_{n-1}, and from this we want to infer the next coefficient h_n. The right-hand side of Eq. (30), $\Lambda_n^{\alpha\beta}$, is known by induction hypothesis. Thus the problem is that of solving a wave equation whose source is given. The point is that this wave equation, instead of being valid everywhere in $\mathbb{R}^3$, is correct only outside the matter ($r > a$), and it makes no sense to solve it by means of the usual retarded integral. Technically speaking, the right-hand side of Eq. (30) is composed of the product of many multipole expansions, which are singular at the origin of the spatial coordinates $r = 0$, and which make the retarded integral divergent at that point. This does not mean that there are no solutions to the wave equation, but simply that the retarded integral does not constitute the appropriate solution in that context.

3.1.1 What one would like to do with the file

The sample document is from an article published by the electronic journal Living Reviews in Relativity [32]. It is typical of many of the articles on this website, which contain advanced mathematics; many others contain images and tables. We would like to make the content of equations, tables, and images searchable. We would like to make the equations semantically meaningful, so that they can be understood by search engines and algebraic manipulation programs.

The text of an article is easy to search for words or phrases. Web search engines index web content and point users at pages that contain the words or phrases that the user wants to search for. But mathematical content is not so easily found. Unless the author surrounds the mathematics with text that contains suitable words, a search engine will not locate the page. Sometimes this would be very unnatural, especially if the search user wants to locate equations containing certain terms or concepts that are not central to the immediate purpose of the equation in its context. We would like to enable suitably enhanced search engines to locate mathematical content.

Equations in Living Reviews are created by their authors in LaTeX. LaTeX contains many ambiguities that make it difficult for a machine or search engine to infer from the display of the equation what the equation really means. For example, a raised superscript following a letter could be a power, an index value, or just part of the name of a variable. Searching for mathematical content means resolving such ambiguities. Equations should contain semantic information.

With such information, it would be possible to map an equation in the journal onto its representation in an algebraic program like Mathematica or Maple. Indeed, it would be possible for a browser to be configured to recognise such an equation and allow the reader to work with it. It would be possible for a suitable search engine to extract all equations with the desired content and put them into a Maple worksheet for the user to manipulate.

Similar content could be indexed for images or tables, which are usually displayed as images in Living Reviews articles. A search should be able to locate images of a certain kind or with a certain content. Tables should carry information about their meaning, not just the meaning of individual entries.

With these goals in mind, we would like to create standards for indicating semantic content in the mathematics that is used in our articles, in images of mathematical concepts, and in tables. We would like to create standard constructs that resolve the ambiguities of LaTeX. We would provide special search methods for accessing this content, either in separate files linked to the main articles or - at a later stage of our work - in metadata fields in XML right in the article. Associated with this data would be a representation of the equation in an unambiguous form, such as OpenMath or content MathML, which would be interpretable by an algebraic program. These data could be extracted from the document or an index and used by search engines, individuals searching the website or article, or even readers to gain access to the content of the inclusions.

3.1.2 What types of content do we need to be representable in MoWGLI?

There are many meaningful symbols in an equation, and many levels of meaning attached to its content. In MoWGLI we will focus on the meanings most directly associated with the theme of Living Reviews, namely content in general relativity, astrophysics, and associated areas. Thus, we will not index for searches any mathematics that is "standard". Instead we will index advanced mathematics, such as differential geometry and tensor calculus, and physical content such as the Riemann tensor, the Planck law, and so on.

Our equations will, however, have completely unambiguous mathematical representations, so that they can be understood by algebraic manipulation programs. The resolution of LaTeX ambiguities will require definition of higher LaTeX commands (macros) that our authors will use to compose the equations. These macros can, when the article is processed, place semantic or content information into indices or metadata fields as desired.

3.2 Sample document 2

We present the formal definition of the n^{th} root of a non-negative real number in Coq and our idea of how it should be presented.

The definition is made in two steps: first, we prove that for every non-negative number c and positive integer n there exists a number x such that $x^n = c$; then we turn this existential statement into a function. The Coq code for this piece of mathematics is given below. Some explanation is in place.

The statement of the lemma reads, in mathematical terms: $\forall n : \mathbb{N}.(0 < n) \rightarrow \forall c : \mathbb{R}.(0 \leq c) \rightarrow \exists x : \mathbb{R}.0 \leq x \wedge x^n = c$. Note that the 'overloading' of symbols we know from mathematics, where we write 0 both for the zero of the natural numbers and for the zero of the reals, is not built in in Coq, where we write `(0)` and `Zero`. One can introduce some overloading (and actually, `Zero` is a strongly overloaded symbol in our library, representing 'the' unit of an arbitrary semi-group), but we haven't done that for our statement of n^{th} roots.

The part immediately after the statement of the lemma, from `Proof` until `Qed` is the 'Coq proof': a list of tactics, entered by the user to interactively guide the Coq system to prove the lemma, which is then stored under the name `nroot`. What is actually stored internally in Coq is not this tactics file (the part from `Proof` until `Qed`) but a so called 'proof-term' that was constructed by Coq using this tactics file. This proof-term is a direct encoding of a (very lengthy and detailed) proof in natural deduction. One can exhibit this proof-term as a term but also as a natural language proof (which is usually not done, because it's very long and detailed, obscuring the real important steps, but that could become feasible once we dispose of a tool to collapse or expand on demand the subproofs). So to be precise, `nroot` is defined as this proof-term.

The `Definition root_fun` actually defines the n^{th} root function: given four parameters, a $n : nat$, a proof of $O < n$ (the argument `posn`), a $c : \mathbb{R}$ and a proof of $0 \leq c$ (the argument `posc`), it defines the real number $\sqrt[n]{c}$ as the x for which $0 \leq x \wedge x^n = c$ holds according to the proof `nroot` of the lemma.

In Coq, comments are put between `(* *)`. This text is skipped by Coq. In the proof and definition of n^{th} roots, many previously defined notions (with their own special syntax) and previously proved properties are used. So actually a mathematical development consists of a series of Coq files (with dependencies between them) in which a mathematical theory is built up.

```
Lemma Set_nrootIR : (n:nat)(lt (0) n)->(c:IR)(Zero[<=]c)->
{x:IR & (Zero [<=] x) * (x[^]n [=] c)}.

Proof.
Intros n n_pos c c_nonneg.
```

```
LetTac p := _X_[^]n[-](_C_ c). (* Auxiliary definition *)
Cut (Monic n p). Intro.
Elim (Set_cpoly_pos' ? p Zero n); Auto.
(* Important theorem application *)
Intro X. Intro H0. Elim H0. Clear H0. Intros H0 H1.
Cut {x:IR & (Zero [<=] x) * ((x [<=] X) * (p!x [=] Zero))}. Intro.
Elim H2. Clear H2. Intro. Intro H2.
Elim H2. Clear H2. Intros H2 H3. Elim H3. Clear H3. Intros.
Exists x. Split. Auto.
Apply cg_inv_unique_2.
Step (_X_!x)[^]n[-](_C_ c)!x.
Step (_X_[^]n)!x[-](_C_ c)!x.
Step_final (_X_[^]n[-](_C_ c))!x.

Apply Set_ivt_poly; Auto. (* Main theorem application *)
Apply monic_apzero with n; Auto.
Unfold p.
Step_leEq_lft (_X_[^]n)!Zero[-](_C_ c)!Zero.
Step_leEq_lft (_X_!Zero)[^]n[-]c.
Step_leEq_lft Zero[^]n[-]c.
Step_leEq_lft Zero[-]c.
Step_leEq_lft [--]c.
Step_leEq_rht [--](Zero::IR). Apply min_resp_leEq. Auto.
Apply less_leEq. Auto.
Unfold p.
Apply Monic_minus with (0).
Apply Degree_le_c_.
Pattern 1 n. Replace n with (mult (1) n).
Apply Monic_nexp.
Apply Monic_x_.
Auto with arith.
Auto.
Qed.

Definition nroot_fun
[n:nat; posn:(lt O n); c:IR; posc:(Zero [<=] c)] : IR :=
(projS1 ? ? (nroot c n posc posn)).
```

THEOREM For every positive integer n and non-negative real number c there exists a real number x such that $x^n = c$.

PROOF Let p be the polynomial defined by

$$p(y) = y^n - c.$$

This polynomial is monic, so we know that there exists a real number X such that $0 \leq p(X)$.

Also, $p(0) \leq 0$, so, by the Intermediate Value Theorem for polynomials, we know that there exists a real number $x \in [0, X]$ such that $p(x) = 0$. This number also satisfies $x^n = c$.

The number x is then called the n^{th}-root of c. QED.

It should be possible to generate an approximation of this text automatically from the Coq source[2], eventually guided by information included in comments in the Coq file. In the example, we have highlighted the important mathematical steps which we want to single out; the other parts of the proof are basically verifications of side conditions. We single out some important high-level aspects.

How do you produce this document? If you want to have completely formal content, you should start from a formal file. So, the document is produced by first creating a Coq file, where the user enters (most likely as Coq comments) information to guide the presentation. This also implies that the (mathematical) text above is seen as a *view* of the formal document (the Coq file with additional presentation information). This view is produced by taking the formal document (with additional presentation information) and generating a printable (ps or pdf) or web document (HTML). In the generation of this view, we see MoWGLI as an intermediate level: first we generate a MoWGLI document, consisting of file in a certain XML language and from this the other documents are generated in a canonical way.

How can MoWGLI **help to write the document?** As stated above, we see the Coq files with presentation information being created interactively using Coq, possibly with an editor/interface like Proof General or PCoq. Communicating with Coq from the nice MoWGLI output (e.g. within a web browser) would be perfect but also far too hard a problem to address right now. What one could use MoWGLI for, while creating the Coq file, are the following two things:

- Search for mathematical content in the library. The search facilities in PCoq are relatively restricted and the ones in emacs are just string searches. A tool exists in Coq to retrieve statements up to isomorphism (for instance up to the commutativity of $A \wedge B$) but more various 'content-based' searching tools would be profitable. String search may be profitable too but needs a discipline and somehow a standard usage on the way to name mathematical notions, properties and lemmas (alternatively: needs to recognize synonym notions, such as symmetrical/commutative, antonym notions, such as lesser/greater-or-equal, abbreviations such as sym for symmetry, order of words such as plus_sym/sym_plus; also: to have both a full description and a key identifier for all notions such as "right associativity of addition on natural numbers"/plus_assoc_l)
- Present existing Coq files in a 'mathematical way' (as indicated above), to make it easier for the user to keep an overview of what's there. This could comprise the file that the user is presently working on, if the rendering can be done on the fly. It should be noted

[2]This proof script is a nice example of how a "good" proof-script should be written by a well-educated Coq user interested in rendering problems. However, the typical user who is usually only concerned with the existence of a proof produces scripts that lack much information that can not be retrieved without the help of the system itself. For example, `Intros H0 H1.` can be simply replaced with `Intros.`, leaving to Coq the burden of choosing the name of the two hypotheses. Understanding those scripts without replaying them in Coq is hardly possible.

that, if one really wants to use the MoWGLI output while creating a Coq file, it should be possible to get access (through the MoWGLI output) to the underlying Coq files, to know the precise names of things, the order in which things are instantiated etc.

How does one use the files? The files should be usable for both didactical and research purposes. In the first case, a different view might be chosen then in the second case. Also, different people might choose a different view, depending on their preferences. As stated above, the files should mainly be usable by (1) other developers who want to build on top of the present Coq theory development (2) students or researchers interested in the subject who want to get an impression of the specific piece of mathematics (3) users interested in understanding all the formal steps of the proof, either because they are implementors of Coq willing to understand the output of the tactics they have written or because an interesting part of the development consists in the formal details (e.g. study of paradoxes in type-theory).

3.2.1 What one would like to do with the files

Going back to the mathematical piece of text, given above, we describe what we would like the MoWGLI representation to allow for.

- Inspection of definitions like "real number" and "monic polynomial". This does not always mean that we want to see the full definition of a notion, because we are often more interested in its (basic) properties instead. For example, we probably don't want to know the definition of real number, but instead see that they form a complete ordered field with the Archimedian property. This gives rise to various levels of detail ('descriptive layers') in which one might want to inspect defined notions.
- Retrieval of information about the lemmas used: what is the statement of the Intermediate Value Theorem for polynomials? How were the free variables instantiated? What side conditions were generated and how were they dealt with?
- Inspection of parts of the reasoning that are hidden, e.g. that every monic polynomial is at some point greater than any given value. Here as well there are various levels of detail in which one can inspect this. These descriptive layers now provide a refinement of the proof.
- Access to other documents which contain relevant information for this piece of mathematics: properties of natural numbers, real numbers, polynomials, monic polynomials; the Intermediate Value Theorem, other variants/corollaries of this theorem…
- The user who is developing proofs may want to look at the Coq script to see how a particular lemma was used. This should be possible both in a frozen and in an interactive way, that is, where the user points/clicks somewhere in the proof and the relevant portion of Coq code is highlighted.
- Keeping also in mind that this is a constructive proof and that Coq has the capacity of extracting programs from proofs, the user should also be allowed to use the program associated to this proof and the corresponding definition; or, alternatively (and perhaps preferably), (s)he should be able to comunicate with a computer algebra system that would output the relevant result.

3.2.2 What types of content do we need to be representable in MoWGLI?

- Formal properties of mathematical notions in various layers of detail. These should be present in such a way that one can easily inspect a deeper layer. For example, a mathematical expression is tagged with an explanation, which again contains mathematical expressions that again have explanatory tags.
- Mathematical proofs on various levels of detail, to be used to generate different views of the document. These consist of textual components and mathematical expressions.
- The two mentioned above require the support to encode text interleaved with mathematical expressions (formulas).
- Meta-information about the notions involved: in what contexts is the information about n^{th}-roots relevant? What other concepts does it require? If a user developing a proof asks for relevant information, when should properties of n^{th}-roots be considered relevant? If a user searching the library for results in specific areas, which keywords should be associated with these lemmas?
- Document-structure, like proofs, examples, remarks but also on a higher level: sections and chapters.

3.3 Sample document 3

3.3.1 Security evaluation of IT products

The fourth application scenario is the evaluation of Information Technology (IT) products following the Common Criteria (CC) standard for security evaluations. The Common Criteria is an ISO standard resulting from the unification of previous European, Canadian and United States standards .

A security evaluation following the CC standard involves three main actors. The Developer is the one who conceived, designed and implemented the IT product. The Developer may ask to start an evaluation procedure of the product to an authorised Evaluator. The Evaluator checks that the IT product meets the requirements that have been specified by the Developer and judges the pertinence of those requirements. Finally, the Evaluator submits a report to the Verification Authority of the country, who checks that the rules of the evaluation process have been respected and decides to deliver or not a security certification for the product. The Common Criteria specifies the documentation to be provided by the Developer to the Evaluator and the points to be checked by the latter [25]; provides a catalog of requirements for security functions [26]; and states different assurance measures on the way of developing programs, testing the product, writing associated documentation, managing configurations, and so on [27].

The Common Criteria considers seven assurance levels, EAL1 being the weakest, and EAL7 the strongest. In level EAL5, the Developer shall include a mathematical description of the security model to be enforced by the product. In level EAL7, the specification and design of the product shall also be modelled. Moreover, a proof of refinement correspondence (RCR) is required between the refinement levels considered in the development of the product (functional specification, high level design, low level design, implementation). Evaluating a IT product may thus involve the development of mathematical models and proofs by the Developer, and its explanation in an appropriate format to the Evaluator.

3.3.2 Documents involved in the evaluation procedure

Usually, the input provided by the Developer to the Evaluator consists in a collection of documents of different kinds.

The Security Target. First of all, the Developer shall provide a Security Target. This document describes the security threats to be countered, the objectives that contribute to counter the specified threats, and the requirements met by the security functions of the product in order to achieve those objectives. An example of a security requirement for an IT product is the enforcement of a security policy. A security policy is a collection of rules specifying the conditions to be fulfilled by the subjects (for instance, users) for executing an operation controlled by the policy on a given piece of data (for instance, whether an user has the right to read the contents of a Unix file). Such rules can be seen as an informal description of a mathematical object, namely, a ternary relation between subjects, controlled operations and accessed objects.

Formal Security models. In high assurance levels (EAL5 to EAL7), the Developer shall produce documents describing the formal security policy model (SM) to be enforced by the product. In Trusted Logic, those models are developed using the Coq proof assistant [28]. Typically, a TSP model consists in a collection of state machines and a set of security properties expressed in terms of those machines. The transition rules of the state machines can be seen as a complete formal account of the security policy rules that are informally described in the Security Target document. Similarly, the invariants are a formalisation of the security objectives fixed in that document. The CC standard requires the Developer to support the formal, mathematical definitions of the model with explanatory, informal prose. The SM documentation hence alternates English paragraphs explaining modelling choices and intended meanings, the display of Coq definitions (of inductive sets and predicates, functions, etc), and references to other IT and CC documents.

The following paragraphs are a sample of that kind of document, taken from the security model developed in Trusted Logic for the Java Card Virtual Machine:

JCVM document sample

Runtime Data Areas

Runtime Data Areas are defined in [JCVM] (section 3.5) as "data areas used during execution of a program". Furthermore, "some of these data areas are created on Java virtual machine start-up and are destroyed only when the Java virtual machine exists". As stated in [JCVM] (section 3.3) : "any runtime data area in the Java Virtual Machine which is duplicated on a per-thread basis will have only one global copy in the Java Card virtual machine".
The runtime data areas of the JCVM are:

- its *execution status*, which indicates whether the JCVM is running or not and if not why it is stopped,
- its *Heap*,
- its static fields (or static variables).

Hence the following formal definition:

```
Record jcvm_state : Set :=
  JCVM_State
  {state_execution_status : jcvm_execution_status;
   state_heap             : jcvm_heap;
   state_field_images     : static_field_images
  }.
```

The next sections analyse in detail each part of the `jcvm_state` structure.

Execution Status

The *execution status* of the JCVM may be one of the following:

- running normally, with an execution state given by a *stack of frames*,
- stopped normally, with an empty *stack of frames* and possibly with a return value,
- stopped with some *uncaught exception*,
- stopped with some fatal error.

The `jcvm_execution_status` type is defined as the following union type:

```
Inductive jcvm_execution_status : Set :=
  Frame_Stack         : jcvm_frame_stack -> jcvm_execution_status
| Return_Value        : (option data)    -> jcvm_execution_status
| Uncaught_Exception  : reference        -> jcvm_execution_status
| JCVM_Fatal_Error    :                     jcvm_execution_status.
```

Program development. In the EAL7 assurance level, the Developer shall also ensure that the security mechanisms can be traced through the different representation levels, and relate the functional specification of each component of the product to the formal security model. This means that the functional specification (FSP) and high level design (HLD) of the software parts of the IT product shall be formally developed. In Trusted Logic, the FSP and HLD levels are also developed using the Coq proof assistant. FSP is presented in the form of pre-conditions (i.e., mathematical properties) to be fulfilled by the input of the programs, and post-conditions (i.e., a mathematical relation) relating them to the output produced by the program. The HLD is described modelling each component of the program as a mathematical function mapping the input to the output of the component.

The contents of this kind of document alternates modelling choices explained as English prose with documentation about the Coq modules describing the HLD of the product, the dependencies between the modules, their interfaces, comments on the Coq code, etc. A possible model that could inspire this kind of document in the MoWGLI project is provided by the JavaDoc tool for documenting Java programs [30]. Some efforts in this direction has been independently carried out by the HELM project at the University of Bologna and by Jean-Christophe Filliatre at INRIA.

Proof documents. EAL7 level also requires to formally prove that HLD functions actually realize the functional specification, that is, a proof of refinement correspondence between the FSP and HLD level. In addition to this, the Developer shall also formally prove what is the specific contribution claimed for each component to the security model. This last kind of document usually alternates English explanations with mathematical proofs like the ones that can be found in a Mathematics or Computer Science article.

3.3.3 What one would like to do with the files

The MoWGLI representation of those documents shall be oriented to the persons embodying the different roles of a security evaluation of an IT product following the Common Criteria standard (Developer, Evaluator, Evaluation Authority).

The documents shall support the idea of an active evaluation of the documents, by opposition of a passive, sequential reading of them, where the Evaluator is mostly limited to just follow the presentation order proposed by the Developer, and focus on what this latter one puts forward or highlights.

An active evaluation shall enable the Evaluator to zoom the level of detail of the definitions and proofs of the document, for instance by unfolding a complex definition of an inductive predicate in Coq, or by displaying the definition associated to a name used in another definition. On the other hand, the Developer of the document shall be able to provide a default view of a huge definition where some parts of it are folded, so that the attention of the Evaluator is paid on the important parts of the definition. Contrary to unstructured documents, this default view should not prevent the Evaluator to display and inspect the whole definition if she likes.

The documents should also help the Evaluator to trace a given functionality from the Security Target to the implementation level through the different refinement levels involved in its development (threats, security objectives, security function requirements, formal security model, functional specification, high level design of the components, etc). Presently, this is done using a sequence of association tables, one mapping each threat to some security objective, a second one mapping each objective to some security function requirement, and so on. The introduction of hyperlinks and structure in the documents should make this technique obsolete.

The documentation shall also enable to search and retrieve information from it more easily. For instance, if the formal security model of the product relays on some non demonstrated Coq axiom, the document should enable the Evaluator to easily find if some important result claimed by the Developer depends on that axiom. Similarly, it should enable to retrieve all the parts of the documentation which are related to a particular security mechanism, from the Security Target document to implementation ones. Remark that the traditional, sequential order of the document is usually oriented by pedagogical purposes, and not for such traversal views of it.

Finally, as a long-term goal, the markings used to describe the mathematical structures included in CC evaluation documents could be also used to communicate between the persons and software tools involved in the different steps of the development of the IT product. This particularly concerns Coq and the tools included in Trusted Logic CCC suite: TL-CC, TL-FIT, TL-CAT. TL-CC is an editor oriented to writing Security Targets and Protection Profiles. TL-FIT is an UML-based CASE tool customised for the development of software compliant with the CC requirements. TL-CAT is a tool for automatic generation of test scripts from a formal specification of the expected product behaviour. In an ideal future environment, it could be expected that, say, the security policy rules included in the Security Target generated by TL-CC could be put in relation with the formal specification to be used by TL-CAT. Similarly, the structure of the data model, described in TL-FIT documents in the form of UML diagrams, could be used as a starting canvas for building a Coq model of it. In all this examples, the documents generated by the tools would be both used as an output for humans and as an intermediate format to communicate between software engineering tools.

3.3.4 What types of content do we need to be representable in MoWGLI?

In principle, three kind of documents are involved in an EAL7 evaluation: CC security targets, documents supporting formal models, documents supporting formal proofs, and documents supporting formal program development.

The contents of the a Security Target have been precised in [25]. A first Document Type Definition (DTD) of that language has been developed by Trusted Logic, but is still to be enriched. In particular, the grammar for describing security policy rules is an issue that could be addressed in MoWGLI.

The basic requirements on contents for documents concerning the support of formal models are very close to the ones concerning the formalisation of Mathematics in Coq (see Section 3.2.2):

- In-place folding and unfolding of Coq terms and definitions. In particular, it would be interesting to be able to display the definition of an inductive predicate and its introduction rules so that only the premises and conclusion of a particular introduction rule are visible, the other introduction rules being folded. The folded rules can be unfolded by the reader if necessary, since the document contains the structure of the whole definition.

- Glossary of all the Coq modules and Coq definitions of the model.

- The various dependency relations between different kinds of Coq entities, like for instance:

    – An identifier and its definition.

  - A mathematical object and its type.
  - All the Coq modules required by the definitions in the document,
  - The lemmata used in a proof (not every constant, but just lemmata).
  - The axioms that a proof (directly or indirectly) relays on.
  - Definitions exported by a Coq module (that is, used in another Coq modules of the model that is being documented).

  Beyond the interest they have in themselves, all these dependencies also provide search domains that can be used in order to simplify information retrieval.

- Specific information concerning CC requirements and the IT product being modelled. Examples of this associations are :

  - The security requirement in the Security Target document that a particular definition is intended to fulfil. For instance, an inductive predicate may be the formal counterpart of the informal rules of a security policy described in the Security Target.
  - The specific CC component (TSP model, FSP, HLD, RCR, etc) associated to a formal definition.
  - The informal source that inspired the definition. Usually, formal models are constructed from an informal description of the IT product, like user manuals, informal specification documents. In that case, it is useful to place a reference to the section or page in the informal document.

  All these cross-related references are intended to provide a traceability of the different functionalities of the product along the documentation.

- As most of the models are based on the use of transition systems and abstract state machines, special support and notation for that kind of mathematical notions could be helpful.

Development oriented documents. Those documents oriented to the functional specification and the high level design of the IT product meet almost the same requirements than those supporting formal models, but they are rather oriented to programmers and implementation specialists. In particular, the syntax of the Coq specification language shall be oriented to that public. For example, a programming language notation for functions, like `short f(short x);{ return x}`, may be preferred to a more traditional mathematical notation for the same object, like $\lambda x : \texttt{short} \cdot x$. Beyond the presentation purposes, it could be useful that the documents records the structure of the program, and not only the Coq term. For instance, some imperative programming language features (like exceptions, or assignment of records) may be encoded in Coq's programming language. Moreover, it is easy to introduce some notation to make this encoding transparent for the user of Coq. It is important that the associated document contains the structure of the program as the user typed it, and not its internal encoding.

The CC standard require for those documents that they clearly describe the structure of the product in terms of *systems* and *sub-systems* (ie, module structure) and *interfaces* of those systems [27, Section 10]. The document shall reflect the structure of those systems and their interfaces.

Refinement correspondence documents. Finally, documents describing proofs of correspondence shall at least contain a clear statement of the correspondence theorems that are intended to be described, a justification of the hypotheses and pre-conditions restricting the cases where the conclusion hold, the axioms or unrefined results the theorem relays on, the relevant lemmata used in its proof, and an explanation of the proof itself in English for a reasonable level of detail.

3.3.5 Some general requirements

As a potential end-user of the tools developed in the MoWGLI project, Trusted Logic expects those tools to be modular and maintainable.

By modular we understand the opposite to having a monolithic, single document: the vision of a document as composed by (potentially reusable) sub-documents should be always enforced. In particular, if a document is automatically generated from some input (an annotated Coq file, for instance), it shall be always possible and easy to compose it with other pieces of text that have been already written (for instance, a general introduction about the product and its features).

By maintainable we mean that software documents are not intended to be written once and forever, but usually evolve in time, accompanying specifications. This is an important difference with respect to "pure" mathematical documents, where a theorem is proven only once. In industry, when a new version of the IT product is delivered, it is important not to restart documentation from scratch. This requirement is particularly important when designing tools that automatically generate (part of) the documentation associated to a product. If a small change in the source models entails a heavy post-generation edition of automatically generated documents, then automatic tools risk not to be usable in practice.

Finally, another important issue is the coherence between the Coq model and its documentation. By coherence we mean that any definition displayed in the documentation must exactly correspond to the definition in the model. This means that, when some definition in the model is modified, so must be done with all the documents where the definition is displayed. Another property related to coherence is the absence of hanged references in the documentation (for instance, to some Coq constant in the model which does not longer exists). Any tool supporting the generation of documentation should take those coherence problems into account.

3.4 Discussion

Every kind of document considered has its peculiar requirements that will probably induce the choice of a domain-specific markup (that a priori does not exclude the possibility of using an extensible common framework). The goal is to provide generic tools for the required functionalities that can smoothly work on any document for which the functionality is meaningful.

For example, the need for a search at the content level for mathematical expressions is shared by the three scenarios. We shall develop a unified way of allowing users to query mathematical databases or libraries, of any kind. Our solution should be general enough and flexible enough to return mathematical equations (in a format chosen by the user), proofs, definitions, whatever. It should basically be an API (application programming interface) that could be implemented in a variety of ways. A web publisher could, for example, provide a graphical search interface that hides the complexity of the query language behind user-friendly buttons and options, in the end composing a query to the search engine. Other websites could do their own implementations that may look very different. Similarly, the way the website does the search in response to the query would be very document dependent. The query language itself can be something the whole community agree on and share.

Another example is the common need for high-quality rendering of presentational expressions derived by the corresponding content markup. The mechanism to obtain the presentation markup from the content markup must be flexible and modular, because the content markup is extensible.

An example of domain specific requirement, instead, is the possibility of interacting with proof-assistants, which, at present, is likely to play no role for the major part of the authors of mathematical documents.

Thus we need a markup generic enough to support the generic functionalities needed by the whole mathematical community without forbidding the domain specific ones. In particular, it should satisfy the following properties:

- It must be extensible, in the sense that it must be possible to encode new notions.
- It must support the operation of *semantic refinement*, which consist in the possibility to associate to every mathematical operator or notion a more refined semantics, possibly specified in a lower-level markup. This naturally leads to the overloading of symbols. For example we can have just one symbol for the implication, whose semantics may be refined in each occurrence accordingly to the nature of the implication (classical, intuitionistic, linear, ...). Semantic refinement is also useful when the notion already has a definition in the language we are considering: in this case semantic refinement may have a foundational meaning. For example we can override the usual definition of the square root of a number as a partial function that associates to x the unique c such that $c^2 = x$ (if it exists) with that of a total function that associates to x and to a proof of $x > 0$ its square root.
- When we map one expression from a markup to another one (for example from content to presentation markup), we need to be able to identify the source fragment that is mapped to some target expression. We can do that associating to those target expressions backward pointers in the source expression. These references have a totally different meaning with respect to the ones we have for semantic refinement. For example, we can associate to the symbol $+$ in the expression $2 + 3$ both a pointer to the content markup element of the sum of two numbers and another pointer to the definition of the sum (semantic refinement).
- Generating presentation markup by means of simple stylesheet applications. It should be relatively easy for a mathematical author (who uses, say TeX) to add presentation information. For this we should create a collection of standard stylesheets that can be fine tuned by the author for specific desires.
- Have text interleaved with mathematical expressions, where parts of the text may refer (via a tag) to mathematical expressions (giving e.g. a precise definition) and a mathematical expression may refer (via a tag) to a piece of text (giving e.g. an intuitive explanation).
- Create a high-level document structure, ranging from 'examples', 'proofs' etcetera to 'sections' and 'chapters'.

- Support the possibility of enhanced content queries. These should factorise out purely presentational distinctions, such as names of bound variables inside an expression. (If we search $\int_a^b \frac{1}{x^2+1} dx$ we should also find $\int_a^b \frac{1}{y^2+1} dy$.)

4 Standard Languages for Mathematical Knowledge Representation

The section is meant to give an overview of the current (XML) “standards” for markup of mathematical documents.

4.1 The Mathematical Markup Language

The Mathematical Markup Language (MathML) [37] is the W3C Standard which provides a specific encoding for mathematical expressions on the Web, capturing both their content and notation structure.

MathML 2.0 consists of approximately 180 markup elements with their attributes, which can be divided into two major categories: the presentation markup and the content markup.

Both MathML presentation and content markup reflects the recursive structure of mathematical expressions. A MathML expression can be regarded as a tree, where an internal node corresponds to a MathML element, the child nodes are the constituent MathML elements of the parent one, and leaves correspond to MathML atomic elements.

Content elements have a default presentation, but MathML provides also mechanisms to associate both encodings in order to specify both the layout and the intended meaning of a mathematical expression.

4.1.1 Presentation elements

One of the main interest of MathML is that it is recommended by W3C as the standard to enable mathematics to be published on the Web, just as HTML does for texts, and it is likely to be supported by most browsers starting from the to be released Mozilla 1.0 and Netscape 7.

Presentation elements express both the two-dimensional layout and the structure of mathematical notation, being also capable of high-quality rendering.

MathML presentation markup consists of about 30 elements which accept over 50 attributes. These elements are roughly divided into two classes.
Token elements, say the leaves of a presentation expression tree, represent identifiers (`mi`), numbers (`mn`), operators (`mo`), and string literals (say text and whitespace).
Layout schemata, say the internal nodes of a presentation expression tree, can have only elements as content and correspond to two-dimensional notational devices. They build an expression specifying how to construct it by means of its subexpressions. There are several classes of layout schemata. One group of elements focuses on more general layout (such as `mrow` for grouping subexpressions), another group is concerned with scripts (such as `multiscripts`) and finally a third group which deals with tables (such as `mtable`).
There are also a few empty elements used only in conjunction with certain layout schemata.

Note that presentation markup, as well as content, is not intended to be edited by hand, but by automatic means or suitable editing tools.

4.1.2 Content elements

The content markup provides an explicit encoding of the *underlying* meaning of a mathematical expression.

MathML provides a base set of content elements for encoding most of the formulas used from Kindergarten to the end of high school and the first two years of college, that is up to a A-Level or Baccalaureate level in Europe.

Content markup consists of about 150 elements accepting a dozen attributes.
The majority of these elements are empty elements corresponding to a wide variety of common mathematical operators, relations and functions[3]. These elements together with token elements representing symbols (`csymbol`), identifiers (`ci`) and numbers (`cn`), correspond to the leaves of a content expression tree.
A group of elements, such as `matrix` and `set`, is used to encode various mathematical data types, and another, important category of content elements such as `apply` are used to build mathematical expressions and also to construct new mathematical objects from others.
Constructing a mathematical expression essentially corresponds to `apply` an operator, predefined by MathML or user-defined via the `csymbol` element, to sub-expressions.
These elements correspond to the internal nodes of a content expression tree.

MathML provides a precise semantics for its content elements relatively to standard mathematical theories. E.g. the `<plus/>` element defaults to the addition on real numbers (a notion not formally defined, actually just an informal reference to a "standard" axiomatisation of real numbers). By use of attributes, a content element can get a different meaning. E.g. attributes `type="integer"` turns `<plus/>` into the addition on integer numbers. A meaning not provided as part of the standard definition of MathML can be given through the `definitionURL` attribute.

The set of mathematical meaning that MathML (version 2.1) provides is limited. For instance, there is no type to express addition on natural numbers or booleans (where addition can be defined as the "logical or"). Content elements are subject to properties, but these properties are sometimes restrictive wrt to the some mathematical field. E.g. the `<plus/>` is assumed to enjoy commutativity, which prohibits its use as the (non commutative) addition on ordinal numbers.

Continuing with the example of `<plus/>`, it happens that it is not possible to use this element to denotes, say, an abstract commutative associative law of an arbitrary set, since it seems there is no way to abstract an MathML expression over an arbitrary algebraic structure (i.e. only already defined structure are available).

MathML is also concerned only by the extensional properties of the operators (e.g. associativity, commutativity, ...) but not on a intensional behaviour as it can only be the case in any construction of the operators in a formal logic. For instance, naturals in the Coq system (which implements the "calculus of inductive constructions" formal system) are defined as the smallest structure build from zero and the successor operation while in the Mizar system (which implements Tarski-Grothendieck set theory), naturals are defined from the empty set and the operation of taking the union of a set with its elements. This forbids to use predefined MathML content elements for the "low-level", intentional definition of notions as elementary as the set of natural numbers or the addition on it.

However this does not forbid to use MathML for what it is good at, that is as an exchange format for well-known standard high school and first two years of college, when the accent is put on the properties and not on the implementation.

Clearly, this abstraction process between the implementation of a notion and the theory

[3]Subject areas covered to some extent in MathML are: arithmetic, algebra, logic and relations, calculus and vector calculus, set theory, sequences and series, elementary classical functions, statistics, linear algebra.

of properties it enjoys is relevant to the formal system. It is up to the specific implementation system (in our case Coq) to link its constructions with the standard informal mathematical notions (the ones concerning MathML) and to provide enough properties of the implementation to justify its connection with the basic mathematical standard theories[4].

4.1.3 Keeping both Content and Presentation

MathML supplies also mechanisms for combining presentation and content markup by embedding one into the other, or by establishing bindings via so called "semantic mappings".

The second kind of mixing is provided by the `semantics` element, which allows to bind presentation and/or content expressions and subexpressions, even different from MathML, in the same document. An interesting use of `semantics` is in connection with cross-references of the XLink protocol [38].

4.2 OpenMath

In contrast to the very rich language of MathML that defines the meaning of extended presentation primitives, the OpenMath standard [5] builds on an extremely simple kernel (mathematical objects represented by content formulae), and adds an extension mechanism, the so-called **content dictionaries**. These are machine-readable specifications of the meaning of the mathematical concepts expressed by the OpenMath symbols. Just like the library mechanism of the `C` programming language, they allow to externalize the definition of extended language concepts. As a consequence, K-12 need not be part of the OpenMath language, but can be defined in a set of content dictionaries (see `http://www.openmath.org/cdfiles/html/core`). Moreover, OpenMath is purely based on content markup.

The central construct of OpenMath is that of an OpenMath object (`OMOBJ`), which has a tree-like representation made up of applications (`OMA`), binding structures (`OMBIND` using `OMBVAR` to tag the bound variables), variables (`OMV`) and symbols (`OMS`). The `OMS` element carries attributes `cd` and `name` attributes. The `name` attribute gives the name of the symbol. The `cd` attribute specifies content dictionary, a document that defines the meaning of a collection of symbols including the one referenced by the `OMS` itself. As variables do not carry a meaning independent of their local content, `OMV` only carries a `name` attribute.

For convenience, OpenMath also provides other basic data types useful in mathematics: `OMI` for integers, `OMB` for byte arrays, `OMSTR` for strings, and `OMF` for floating point numbers, and finally `OME` for errors. Just like MathML, OpenMath offers an element for annotating (parts of) formulae with external information (e.g. MathML or LaTeX presentation): the `OMATTR`[5] element, which pairs an OpenMath object with an attribute-value list. To attribute an OpenMath object, it is embedded as the second child in an `OMATTR` element. The attribute-value list is specified by children of the `OMATP` element, which is the first child, and has an even number of children: children at even position must be `OMS` (specifying the attribute), and children at odd positions are the values of the attributes given by their immediately preceding siblings.

[4]which does not exclude more marginal alternative to standard theories, such as so-called non-standard analysis or constructive analysis (which by the way is actively developed in Coq)

[5]Note that the meaning of this element is somewhat underdefined, it is stated in the standard, that any OpenMath compliant application is free to disregard attributions (so they do not have a meaning), but in practice, they are often used for specifying e.g. type information.

The content dictionaries that make up the extension mechanism provided in OpenMath are tied into the object representation by the `cd` attribute of the `OMS` element that specifies the defining content dictionary.

OpenMath and MathML are well-integrated:

- the core content dictionaries of OpenMath mirror the MathML constructs (see `http://www.openmath.org/cdfiles/html/core`); there are converters between the two formats.
- MathML supports the `semantics` element, that can be used to annotate MathML presentations of mathematical objects with their OpenMath encoding. Analogously, OpenMath supports the `presentation` symbol in the `OMATTR` element, that can be used for annotating with MathML presentation.
- OpenMath can provide a semantic encoding format for MathML beyond K-12 mathematics: MathML content supports the `csymbol` element, which has an attribute `definitionURL` that points to a document that defines the meaning of the symbol and that can be an OpenMath CD. The content of the `csymbol` element is MathML presentation markup for the symbol.

4.3 OMDoc

OMDoc extends the MathML and OpenMath standards to encompass mathematical documents (see [3]). This is suitable for MoWGLI since almost all of mathematics (specifications and properties of mathematical objects) is currently communicated in document form (publications, letters, e-mails, talks,. . .). As these documents have a complex structure of their own (often left implicit by typographic conventions), the specific task to be solved in the extension to OpenMath is to provide a standardised infrastructure for this as well.

As a consequence, OMDoc provides two sorts of markup devices; for

microstructure of mathematical texts this largely comprises the general pattern "definition, theorem, proof" that has long been considered paradigmatic of mathematical documents like textbooks and papers. Furthermore OMDoc supports auxiliary items like explanatory text, cross-references, exercises, applets, etc. See [3] for details. In a nutshell, OMDoc uses specialised XML elements for all of these which may contain text representations (in form of `CMP`s) and formal versions (in the form of `FMP`s) of the mathematical content.)

macrostructure in terms of mathematical theories. For this, OMDoc techniques from the field of software engineering (see e.g. [17] for an introduction to algebraic specification), which focuses around the structured specification of structured formal theories of the behaviour of software and hardware.

4.3.1 Mathematical Theories in OMDoc

Traditionally, mathematical knowledge has been partitioned into so-called **theories**, often centred around certain mathematical objects like groups, fields, or vector spaces. Theories have been formalised as collections of

- **signature declaration** (the symbols used in a particular theory, together with optional typing information).

```
<theory id="monoid">...
 <symbol id="monoid">
  <commonname xml:lang="en">monoid</commonname>
  <commonname xml:lang="de">Monoid</commonname>
  <commonname xml:lang="it">monoide</commonname>
  <signature system="simply-typed">
     set[any] -> (any -> any -> any) -> any -> bool
  </signature>
 </symbol>...
</theory>
```

Figure 1: An OMDoc symbol declaration

- **axiom** (the logical laws defining the theory).
- **theorem**; these are in fact logically redundant, since they are entailed by the axioms.

In software engineering a closely related concept is known under the label of an (algebraic) **specification**, that is used to specify the intended behaviour of programs. There, the concept of a theory (specification) is much more elaborate to support the structured development of specifications. Without this structure, real world specifications become unwieldy and unmanageable.

OMDoc supports this structured specification of theories; it builds upon the technical notion of a **development graph** [12], since this supplies a simple set of primitives for structured specifications and also supports the management of theory change. Furthermore, it is logically equivalent to a large fragment of the emerging CASL standard [7] for algebraic specification (see [2]).

All specification languages support mechanisms for specifying signature and axiom information, in particular, most also support **abstract data type** as a convenient shorthand for sets of inductively defined objects and recursive functions on these. We will subsume these under the label of simple theories and discuss their representation in OMDoc in the next section. After that we will use section 4.3.3 to discuss the issue of structuring and reusing theories by importing material from other theories.

4.3.2 Simple Theories

Theories are specified by the `theory` element in OMDoc. Since signature and axiom information are particular to a given theory, the `symbol`, `definition`, and `axiom` elements must be contained in a theory as sub-elements.

`symbol` This element specifies the symbols for mathematical concepts, such as 1 for the natural number "one", + for addition, = for equality, or `group` for the property of being a group. The `symbol` element has an `id` attribute which uniquely identifies it (in an OMDoc document).

This information is sufficient to allow referring back to this symbol as an OpenMath symbol. For instance the symbol declaration in 1 gives rise to an OpenMath symbol that can be referenced as `<OMS cd="monoid" name="monoid"/>`.

If the document containing this `symbol` element is stored in a data base system, the OpenMath symbol could be looked up by its common name. The type information

```
<theory id="group">
 <imports id="group.import" from="monoid" type="global"/>
 <axiom><CMP> Every object in
  <OMOBJ><OMS cd="monoid" name="set"/></OMOBJ> has an inverse.
 </CMP></axiom>
</theory>
```

Figure 2: A theory of groups based on that of monoids

specified in the `signature` element characterises a monoid as a three-place predicate (taking as arguments the base set, the operation and a neutral element).

`definition` Definitions give meanings to (groups of) symbols (declared in `symbol` elements elsewhere) in terms of already defined ones. For example the number 1 can be defined as the successor of 0 (specified by the Peano axioms). Addition is usually defined recursively, etc.

For a description of abstract data types see [15].

4.3.3 Complex Theories and Inheritance

Not all definitions and axioms need to be explicitly stated in a theory; they can be inherited from other theories, possibly transported by signature morphism. The inheritance information is stated in an `imports` element.

`imports` This element has a `from` attribute, which specifies the theory which exports the formulae.

For instance, given a theory of monoids using the symbols `set`, `op`, `neut` (and `axiom` elements stating the associativity, closure, and neutral-element axioms of monoids), a theory of groups can be given by the theory definition using `import` (see Fig. 2).

`morphism` The morphism is a recursively defined function (it is given as a set of recursive equations using the `requation` element, described above). It allows to carry out the import of specifications modulo a certain renaming. With this, we can e.g. define a theory of rings given as a tuples $(R, +, 0, -, *, 1)$ by importing from a group $(M, \circ, e, i)$ via the morphism $\{M \mapsto R, \circ \mapsto +, e \mapsto 0, i \mapsto -\}$ and from a monoid $(M, \circ, e)$ via the morphism $\{M \mapsto R^*, \circ \mapsto *, e \mapsto 1\}$, where R^* is R without 0 (as defined in the theory of monoids).

5 Authoring

MoWGLI documents will exhibit a strong need for internal consistency, because the same concepts will appear at different levels: for instance, mathematical operators appear as functions and links to the definitions of these functions may be provided. If a definition is provided, this definition may state whether the operator is binary, ternary or n-ary. Specific notations for these operators may also appear in stylesheets or in transformation descriptions given in XSLT. Producing this kind of documents while ensuring the internal consistency will become a cumbersome task, unless tools are provided to reduce the amount of information that authors will need to provide by hand and to compute data such as paths for cross link references.

The scope of the MoWGLI project is already large enough for us to consider several categories of tools depending on the context in which the authors work. A specific domain for which available internal consistency will be especially dense is the domain of formalised mathematics with the help of a proof assistant, where the proof assistant will be used to check the consistency of documents up to a very precise logical consistency. We discuss this domain in a first section. The more general working context corresponds to situations where mathematical formulas are manipulated symbolically with computer algebra systems to help produce correct formulas more quickly, but justifications are given with informal text. We discuss the tools for this kind of documents in a second section.

5.1 Authoring in the context of proof assistants

When working with a proof assistant, authors of mathematical documents are led to produce documents where a lot of details are provided and formally (i.e., mechanically) checked. This formal documents are usually very verbose and yet very terse and a lot of work is needed to make them suitable for human reading. In fact we can envision two kinds of activities in this context, depending on whether the authoring tool is used to create new mathematics or formalise mathematical proofs that were already studied in an informal way (what we could call pen-and-paper proofs), or to construct a MoWGLI document from an already complete formalised proof. Of course, the boundaries between these kinds of activities can be blurred, but each activity will make it possible to underline a certain category of tools.

5.1.1 Creating new mathematics

The activity of creating new MoWGLI documents should be compared to that of creating new software, and authoring tools should provide functionalities that are comparable to those of software engineering tools.

In particular, authors should have the possibility to sketch the structure of their mathematical developments in a manner that makes it possible to foresee the main shape of the final MoWGLI documents. It should be possible to give central concepts and theorems only by name and informal description first, with the subsequent work concentrating on raising the level of details and checking the consistency between the requirements and results of each objects. Requirements for a definition could be that a certain element should be defined uniquely, requirements for a theorem correspond to its premises.

As the process of raising the level of details progresses, it may be necessary to re-consider previous design decisions, as one discovers that some theorem cannot be proved under some set of assumptions or as one discovers that the results being proved are more powerful than initially foreseen. In these conditions, it should be possible to adapt previous designs in various ways: evolutionary steps will not all correspond to raising the level of formal detail and sometimes fragments of correct, but useless, mathematical documents will have to be dropped to achieve a better design. All decisions should be recorded in a way that they can be backtracked gracefully, if possible without abandoning unrelated design decisions that may have occurred later. In all this process, the aim to produce human readable documents as well as machine readable documents should also be kept in mind and in particular the main design and the various elements (definitions and theorems) should be annotated as the design progresses with some informal text (or enriched text, since it should be possible to have mathematical notations in the middle of the text).

When authors start from a pen-and-paper proof, they are more likely to have a clear understanding of the main structure of the proof. The need to re-consider the design appears less often. Still, the capability to associate informal explanations from the paper proof to the electronic documents should be made available in a strong way.

5.1.2 Putting Formal Mathematics on the Web

Over the last thirty years, semi-automatic proof assistant have developed a mode of operation where commands are sent one-by-one to a proof assistant that produces some output at each step to indicate to the user the way definitions are accepted and proofs advance. The results of this kind of activity usually is a file where the commands produced by the user are recorded, but not the proof assistant's output. It is usually easy to check that this file is consistent, simply by re-running the file through the proof assistant's command line interface.

In some sense, the proof process is a dialog between a clever actor (the human user) and a stupid one (the machine) and the file only records the contribution of the clever actor. The part of the stupid actor is predictable and does not need to be recorded. Still, only a machine can predict that part, because this process usually requires more memory than what is available to the common human reader.

When constructing MoWGLI documents, some of the missing information needs to be re-produced and this activity should be monitored by the human author by adding indications to the session file that make it possible to produce a new file containing information coming both from the session file and the corresponding proof assistant's output.

In the current state of the art, tools are provided by the proof assistant developers to perform this task, for instance with the tool called `coq-tex`. This tool actually considers the user-provided input file as a LaTeX source, i.e., a file principally designed to produce a typesetted document, from which a session file is extracted and the answers to some commands (earmarked by the user) are inserted in the initial LaTeX source.

An orthogonal approach is provided by another tool, named `coqdoc` [11], where the user-provided input file still is viewed as a session file, but special comments are used to insert typesetting or structuring instructions, that can then be used to generate either paper documents (using LaTeX) or a hypertext document. When a hypertext document is created, links are automatically added to relate identifier usages to definitions. However, no directives are provided to make sure that some of the proof assistant's output appears in the final document.

None of the existing tools actually support a conversion to realistic mathematical notations: the text that is meant to serve as input to the proof assistant appears as plain text. Therefore a first simple requirement is that there should exist a tool that combines the capabilities of `coq-tex` and `coqdoc`: include, upon request, outputs from the proof assistant, provide links from usage to definition; and add realistic mathematical notations.

However, only using session files to describe a mathematical development is insufficient to produce truly readable documents, since users tend to insert in these session files only the final result of a mathematical proof developments and main structure of the theory underlying this development, the links between various concepts are missing in this file. Several ways, can be provided to recover the structure in a development, very often with the help of systematic tools to compute the dependencies between various commands or concepts.

The distinction between “natural language” text as informal, user-provided annotation to some “machine language” text as formal text used as machine input or produced as machine output should also be blurred by the new capabilities related to natural language processing. Some of the “natural language” text can actually be produced automatically by the proof assistant and natural language can also be used as machine input, thanks to the recent progress observed in the use of natural language as input language for logical tools.

5.2 Authoring in a free context

Working with a proof assistant leads to very precise descriptions of mathematics, but the cost can be discouragingly high. Authoring tools should also support the production of MoWGLI documents where less verifications on the logical consistency of mathematics are performed. In this context, the MoWGLI documents that are being produced still require a large quantity of work for which automated support can be important. One aspect revolves around the editing of mathematical formulas, that will be discussed in Sect. 8.6; the other relates to the task of importing inside MoWGLI documents data produced by other mathematical tools. One important class of these tools are symbolic computation systems, that play about the same role as the proof assistants in previous sections. In that case, the directives given by the user correspond more to the commands to be sent to the logical system to make sure it produces the relevant data. This data comes in a form that is closer to semantic data and transformations need to be performed to produce regular content-based descriptions and presentation descriptions. Here, because data is not provided manually, it is absolutely not possible to consider that the user should have control on where line breaks should occur in formulas, because formulas are not part of the user-provided input. This characteristic underlines the way user guidance for the generation of presentation data should be provided.

For example, symbolic computation tools should include Mathematica and Maple which, both, contain sufficient space to write complete documents. Advantage should be taken from their (partial) support for MathML.

6 Stylesheets and Transformation

An important part of the descriptive power of mathematics derives from its ability to represent formal concepts in a highly evolved, two-dimensional system of symbolic notations. Tools for the mechanisation of mathematics and the automation of formal reasoning must eventually face the problem of re-mathematization of the logical, symbolic content of the information, especially in view of their integration with the World Wide Web. We have already discussed the pivotal role that XML-technology is likely to play in such an integration, and the new potentialities offered by languages such as MathML [37] for rendering mathematical notation on the Web. In this section, we focus on the problem of (Web) publishing, advocating the use of XSLT stylesheets as a standard, application independent and modular way for associating notation to formal content.

The Extensible Stylesheet Language, whose Transformation part (XSLT) [39] has recently become a W3C recommendation is a simple rule-based language for transforming XML documents into other XML documents. Although the expressive power of XSLT is remarkable, it was not intended as a completely general-purpose XML transformation language but, primarily, as a mean to specify the *styling* of an XML document by transforming the specific XML-dialect of the input document into a formatting language suitable for rendering issues (HTML, Formatting Objects, or whatever). In MoWGLI, we plan to extensively use XSLT as a standard mechanism for associating notation to content in mathematical documents (see [1]). The broad goal, here, is that of developing coherent and well maintained libraries of notational stylesheets, publicly available on the web and freely reusable by any interested party.

Stylesheets have been already successfully used for rendering mathematical content by the OpenMath community. Among others XSL stylesheets for the display of OpenMath documents in MathML presentation, in plain-html and in TeX are available. The fact that this rendering is performed using XSL stylesheets makes it extensible and easily modifiable (the simplicity of the XSL template example in Figure 8 applies here as well) (see for example the OMDoc or ActiveMath stylesheets for html and LaTeX). One of the important purpose of these modifications are the enrichment with interactivity features such as, in a Web-browser, the display in the status bar of the name of the symbol being flied over by the mouse. Moreover, other presentation mechanisms exist, which may perform more efficiently; for example the RIACA group of the Technical University of Eindhoven have implemented a rendering to images and a rendering to MathML.

The transformation of a document from the internal representation in some system to its final rendering essentially goes through four phases: exportation, transformation, presentation, and rendering. Fig. 3 provides the overall simplified architecture of these phases.

Rendering will be discussed in Section 7, together with the tools required for the management of stylesheets. In this section we focus on the first three phases.

The only phase that is application-dependent is the first one. The second phase may depend on the specific foundational framework used by the application (for proofs and, to a certain extent, also for statements), but it is already decoupled from the specific application. The third phase is quite general and can be shared by most systems (especially for the notational support, which is the most prominent aspect of this phase).

As shown in Fig. 3, there are two main flows of transformations, according to the two possible outputs, namely theories or individual objects. With theory we mean a collection of objects, possibly intermixed with text and images, structured into sections, chapters and so

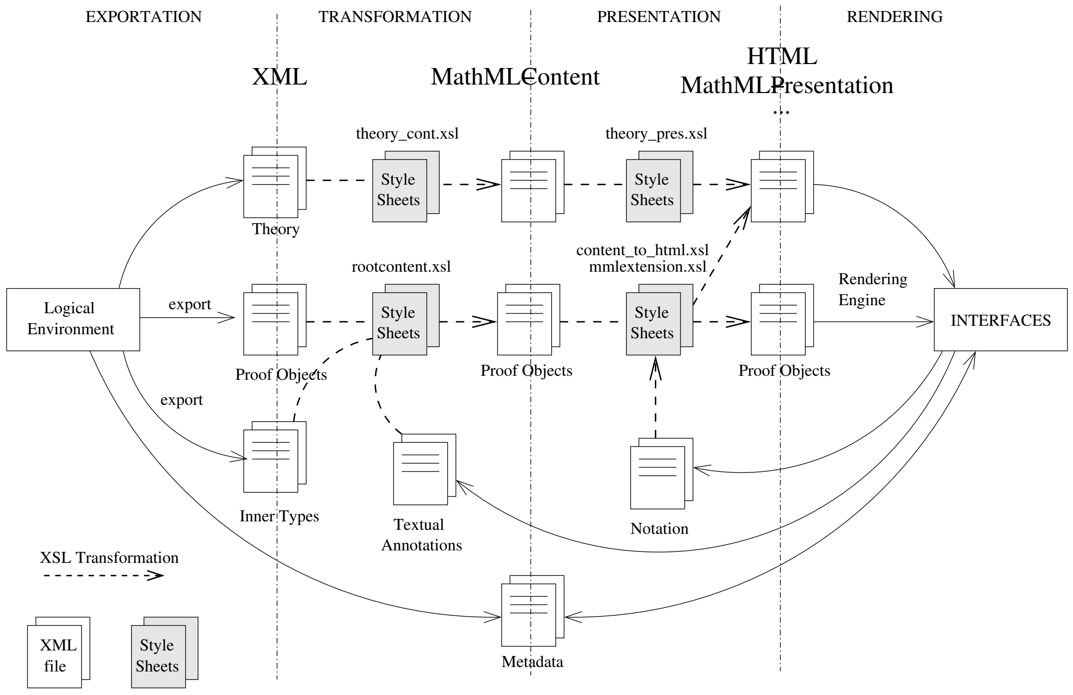

Figure 3: Transformation Phases and main stylesheets

on. The most complex transformation process obviously concerns objects (and its sublevels: namely propositions, terms and proofs, comprising the required notational support). The interesting problem regarding theories is their dynamic generation (see Sect. 8.3); once created, they do not require major transformations, relying on the transformations of the embedded objects.

We shall now discuss some of the most crucial aspects of the first three phases.

6.1 Export

In the first phase the document is exported to XML. Accordingly to the nature of the system, the information requires a specialised semantics encoding or can use some standard content encoding. In particular, Computer Algebra Systems are likely to be able to import and export to MathML Content or OpenMath and often to use this media as a meaningful communication format. Indeed, the development of the OpenMath standard went together with the development of specification and implementation of *phrasebooks*: wrappers for mathematical systems which can receive and send OpenMath expressions, possibly enriched with system-specific commands. Examples of such systems have been developed by WebPearls Inc., by the PolyMath group in Simon Fraser University (for the Maple computer Algebra system), by the RIACA group (for the Mathematica and GAP computer algebra systems), and by others. Nevertheless, experiments with these systems prove that a content-standard that would reflect faithfully all mathematical systems is barely possible (this applies to both OpenMath and MathML-content): Maple and Mathematica, for example, disagree on many definitions of functions branch-points or the floating point algorithms used.

Proof assistants, instead, really requires a semantics encoding that typically makes sense only for a specific logical system. Embedding the semantics markup into a meta-language as OpenMath would generate several unrelated content-dictionaries, one for every logical framework, without providing any added value in terms of sharing and communication of data.

In what follows we will focus our attention on exporting information from proof-assistants. In MoWGLI, this is conceived as a batch process, producing the actual and persistent library of mathematical documents.

The information encoded in markup should exactly reflect *all* the internal information of the application, and not just the information required for rendering purposes. For instance, this layer could be profitably used as a more convenient format for exchanging formal pieces of the library on the Web.

The necessity of having a specific description is motivated by the fact that the encoded information is very different both in content and format from one system to another. A general purpose language would hardly reflect all the small but essential details of the specific internal representation of the information within a given logical framework. For instance, Fig. 4 provides an example of a natural low-level encoding in XML of the expression $(n > O) \vee (O = n)$ for the Coq Proof Assistant.

```
<APPLY>
 <MUTIND notype="0" uri="cic:/coq/INIT/Logic/Disjunction/or.ind"/>
 <APPLY>
  <CONST uri="cic:/coq/INIT/Peano/gt.con"/>
  <REL binder="n" value="1"/>
  <MUTCONSTRUCT noconstr="1" notype="0" uri="cic:/coq/INIT/Datatypes/nat.ind"/>
 </APPLY>
 <APPLY>
  <MUTIND notype="0" uri="cic:/coq/INIT/Logic/Equality/eq.ind"/>
  <MUTIND notype="0" uri="cic:/coq/INIT/Datatypes/nat.ind"/>
  <MUTCONSTRUCT noconstr="1" notype="0" uri="cic:/coq/INIT/Datatypes/nat.ind"/>
  <REL binder="n" value="1"/>
 </APPLY>
</APPLY>
```

Figure 4: Example of semantic encoding of the expression $(n > O) \vee (O = n)$ for Coq

The main issues in this phase are to decide which information is worth exporting, to choose the right granularity of XML documents, and the actual definition of the Document Type Definition.

An additional problem of this phase is that some of the information required for presentational issues may not be directly available in the internal representation of the application. For instance, in type-theoretical tools encoding proofs as lambda-terms via the Curry-Howard analogy, the type of the inner nodes of the proof (which are essential to recover a human-readable representation of the proof) is typically missing. Thus, while we export the information, a tight interaction with the application is usually required.

6.2 Transformation

The second phase is the core of the transformation process. In this phase the document is transformed, by means of stylesheets, into a suitable intermediate content representation (a pointer to the formal content is preserved as an XLink [38]). This intermediate level is meant to improve the modularity of the whole architecture. Many different formal notions, from the same or even different logical environments, are typically mapped here into the same intermediate notion. Take for instance the definition of equality, or that of an order relation: their formal definition may be very different from one system to another (or from a sub-theory to another), but their intended presentation (and intuitive meaning) is the same. Therefore, there is no point in defining a specific presentation for each formal notion and we just define presentation for the intermediate content level, mapping all formal notions into the corresponding content one, with the intended representation. MathML content looks here as a promising candidate for the representation of formulae and proofs at this layer; the typical representation of expression $(n > O) \vee (O = n)$ in this language is described in Fig.5

```
<apply>
  <or/>
  <apply>
    <gt/>
    <ci>n</ci>
    <ci>O</ci>
  </apply>
  <apply>
    <eq/>
    <ci>n</ci>
    <ci>O</ci>
  </apply>
</apply>
```

Figure 5: MathML content encoding of the expression $(n > O) \vee (O = n)$

Note the structural similarity between this representation and that in Fig.4. The similarity can be enforced by the fact all MathML content elements accept a `definitionURL` as an attribute (a fact which is essential when MathML is used as an intermediate language, in order to preserve a link to the low level, formal representation). For instance, the greater-than relation is not a primitive notion in Coq: actually, it is a suitable inductive definition. However, when we pass from Coq to MathML there is no point to preserve this information, if not as a pointer to its actual definition. So, the application of Coq-gt constant to its pair of arguments can be directly transformed into an application of MathML-gt element to (the result of the transformation on) its arguments, automatically recovering the intended presentation. The templates in Fig.6 captures this idea.

In a similar way, content can be associated to all user defined notions. In case the notion is not in the primitive set of MathML content, it is enough to create a new `csymbol`, and extend the presentation stylesheet with the intended presentation.

We shall also eventually need to develop suitable extensions for the theory and metadata level (these are really complex tasks, whose discussion will be better detailed in the Report

```
<xsl:template match="APPLY[CONST[attribute::uri='cic:/coq/INIT/Peano/gt.con']
                             and (count(child::*) = 3)]" >
    <m:apply>
    <m:gt definitionURL="{CONST/@uri}"/>
     <xsl:apply-templates select="*[2]"/>
     <xsl:apply-templates select="*[3]"/>
    </m:apply>
</xsl:template>
```

Figure 6: Example of a XSLT template from semantics to content

on "Structure and Metastructure of Mathematical documents"). Moreover, during the transformation phase we must also heavily rework proofs in order to put them in a form more suitable to human reading. Typically, this requires a major reorganisation of the structure of the proof (see e.g. [9, 8]): in a proof assistant, proofs are typically generated in a bottom-up fashion, while we naturally expect a top-down presentation, where subterms (sub-proofs) appear before conclusions. Another complex issue is that of recognising and managing induction principles (one of the main proof mechanisms of constructive mathematics and proof assistant applications). During this phase, proofs must be also typically integrated with redundant information (such as the intermediate conclusions of the proof-tree, see section 6.1), such as to obtain a human-readable output.

A different approach has been followed by Caprotti, Geuvers and Oostdijk [22, 6], generating the natural language rendering off-line, using a tool written in Java; the output is then included once and for all inside the mathematical document (that is an OMDoc instance). Even if the initial implementative effort is surely greater, the MoWGLI approach has clear advantages in terms of user-configurability: changing the natural language output, for example associating a particular verbalization to an operator or changing the whole output to another language, just amounts to adding or overriding a few stylesheet templates. Moreover, this can be done on-the-fly and on a per-user basis.

6.3 Presentation

In the third phase the document is transformed from its intermediate, content representation to the final presentation format. The representation of $(n > O) \vee (O = n)$ in the MathML presentation language is described in Fig.7.

Several stylesheets covering the transformation from MathML content to MathML presentation exist already. For instance, in Fig.8 you may see a fragment of a stylesheet mapping MathML content to MathML presentation due to Igor Rodionov, of the Computer Science Department of the University of Western Ontario, London, Canada (the code has been slightly simplified, for the sake of clarity).

The first template matches any `apply` element which has an `or` element as first child. The template expects to receive in input a parameter called `IN_PREC` expressing the precedence of the context (initialised to the variable `NO_PREC` in case no actual parameter is passed). Then, two cases are possible, according to the relative precedence of the context w.r.t. the precedence of operator `or` (defined by the variable `OR_PREC`). If `IN_PREC` > `OR_PREC` we must first insert a pair of brackets (`<m:mfenced separators="">`) using "blanks" as separators,

```
<mrow>
  <mrow>
     <mi>n</mi>
     <mo>&gt;</mo>
     <mi>O<mi>
  </m:mrow>
  <mo>&vee;<mo>
  <mrow>
     <mi>O<mi>
     <mo>=<mo>
     <mi>n</mi>
  </mrow>
</mrow>
```

Figure 7: The expression $(n > O) \vee (O = n)$ in MathML presentation

and then recursively apply the templates on the same input fragment (`select="."`), entering a special mode called “or”. In the other case, we do not need brackets, and just recursively apply the templates.

The second template is active only in the special mode “or”. We first recursively apply the templates on the second child of the application (i.e. the first argument of the operator `or`) passing `OR_PREC` as current precedence of the context, and then for each other child at position > 2 we recursively apply the templates on it, generating `<m:mo>&vee;</m:mo>` in front of each of them (the `or` operator of MathML takes any number of arguments).

When extending MathML content by means of user defined `csymbols`, we have just to extend the previous stylesheet to cover the new cases (and XSLT has a clean inclusion mechanism to deal with this kind of extensions).

6.4 Stylesheet editing

We will probably need to provide authors with tools to edit the stylesheets used in the transformation and presentation phases. In particular, most of these stylesheets have a very simple and repetitive structure, opening the possibility of automatically generating them from a more abstract and concise representation of notational and stylistic information (see [1, 19]). In fact, the algebraic tradition of mathematics leads to mathematical notations based on few operators with simple typesetting conventions: infix binary operators like addition or subtraction, superpositions. For a specific domain of mathematics, it is often the case that authors need to add an operator, simply using for this operator a new symbol, but display conventions that mimic that of standard operators.

A similar functionality is already implemented in the Pcoq tool, a graphical interface to the Coq proof assistant. In Pcoq a table of operators can easily be edited by the user to add or modify the display conventions associated to a new operator. The graphical user interface provides a quick way to update the display notations. The resulting tables, however, are not organised in a way that makes them specific to a given mathematical document and it does not rely on standard technology such as XML. This should be provided in the context of MoWGLI.

```
<xsl:template match = "m:apply[m:or[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $OR_PREC">
      <m:mfenced separators="">
        <xsl:apply-templates select="." mode="or"/>
      </m:mfenced>
    </xsl:when>
    <xsl:otherwise>
      <m:mrow>
        <xsl:apply-templates select="." mode="or"/>
      </m:mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "m:apply[m:or[1]]" mode="or">
  <xsl:apply-templates select="*[2]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$OR_PREC"/>
  </xsl:apply-templates>
  <xsl:for-each select = "*[position()>2]">
    <m:mo>&vee;</m:mo>
    <xsl:apply-templates select="." mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$OR_PREC"/>
    </xsl:apply-templates>
  </xsl:for-each>
</xsl:template>
```

Figure 8: From MathML content to MathML presentation

For some specific operations, mathematical notations have a complexity that goes beyond the available capabilities provided by a simple table mechanism. In these cases, authors need to produce XSLT template rules, providing pattern information and presentation information corresponding to the notations they want to adhere to. Pcoq contains the necessary building blocks to specify the information and to generate the XSLT rule automatically [10]. Specific support for this activity could involve automatic generation of the pattern part of XSLT template rules from examples of data representing instances of the new operator and interactive tools to help constructing the produced text for most frequent styles of layout.

It should be considered whether the expressivity of the OMDoc `presentation` tags may be sufficient for these purposes.

7 Rendering tools

We now face the problem of rendering mathematical documents on the Web. In the following section we provide a quick overview of the presentation languages of interest in this context. In the following one we discuss the state of the art of MathML presentation engines. Finally we outline the main requirements for an adequate rendering system for MoWGLI.

7.1 Languages for Scientific Documents

7.1.1 XML-based Languages

The languages in this section have been developed by W3C working groups with the explicit purpose of providing publishing technologies for the Web. A common characteristic is the explicit encoding of structure as part of the information conveyed by the document.

- HTML [40] (HyperText Markup Language) is by far the most popular language for Web page description;
- FO [41] (Formatting Objects) are part of the XSLT specification. They are meant to provide a generic publishing mechanism for XML documents that is independent of the rendering device;
- MathML presentation [37] is a rich markup language for the description of the layout of mathematical formulas;
- SVG [42] (Scalable Vector Graphics) provides a language for the description of diagrams and complex graphics, possibly supporting animated content and simple interactive capabilities.

HTML and FO offer very poor support for mathematics by themselves. In fact, one of the key features of these markup languages is that they allow embedding of so called *foreign objects*. For example, it is natural for an HTML document to embed SVG or MathML markup. SVG, in turn, can embed foreign markup, giving the possibility of creating diagrams with HTML or MathML annotations and labels. A recent working draft published by the W3C defines a combined DTD for compound documents using HTML, SVG and MathML [43].

7.1.2 Other Markup-based Languages

The most relevant language in this category is TeX [13], which is by far the most successful language for typesetting scientific documents in academics. Other languages for typesetting, like `troff`, `eqn` and `texinfo` are either obsolete, or they are implemented by compilation into TeX based markup, and can thus be considered equivalent to TeX with respect to their relevance in the context of MoWGLI.

TeX has been explicitly designed to produce high quality documents on paper; it focuses on presentational aspects only. These characteristics make TeX (and its macro packages such as LaTeX) unsuitable for many crucial aspects in MoWGLI, where documents

- are supposed to be cross-linked on the Web;
- convey semantical information either directly (by inclusion) or indirectly (by reference using some machine-understandable mechanism, such as hyperlinks);

- must be deliverable to a variety of rendering devices whose capabilities (resource availability, rendering quality, support for interactivity) span a wide range.

Nevertheless, TEX can play an important role in the project. First of all it can be seen as an effective and mature target language for rendering documents in a specific setting. Furthermore, TEX is a well-established publication language in many conferences and journals. The possibility of exporting information from the library for inclusion into TEX documents is particularly important, until alternative technologies for high-quality publication of documents are available. Finally, XML markup has always been advertised as being human-readable, but its direct editing is a very tedious task. Markup languages like TEX proved to be sufficiently friendly so that they could serve, by proper translation and processing, as concrete syntax for XML documents.

7.1.3 Low-level, Page-description Languages

PostScript [33] and PDF [34] are two very common languages suitable to deliver documents in a "final form", usually for printing purposes. They are mostly used as the result of a rendering process from TEX or XML documents.

PDF, which builds on top of PostScript, has a notion of "object" that enriches documents with some structure. It also has few, limited interactive capabilities.

The two languages are, however, very much oriented towards a particular medium (the paper), they are not Web-friendly (even though recently some search engines offer support for searching inside such documents) and thus not suitable for on-line navigation inside large repositories of mathematical knowledge.

7.2 Applications for MathML

If we look at the current possibilities for rendering MathML markup, applications fall into three main categories, described in the following sections[6].

7.2.1 Web Browsers

The two Web browsers that currently recognise MathML markup natively are Amaya (`http://www.w3.org/Amaya/`), developed by the W3C, and Mozilla (`http://www.mozilla.org`), whose development started after the release of the source code of the Netscape Navigator browser. Both browsers are available for a number of different platforms, in particular Microsoft Windows, MacOS 9 and Unix/Linux/MacOS-X. Microsoft has stated no interest to include MathML support in Internet Explorer, at least in the short term.

For a number of reasons, ranging from irrelevant commercial visibility to the lack of an appealing user interface, Amaya is not a popular Web browser. In general, it is more regarded as a prototype for the implementation of W3C technologies and its support for MathML is limited in both extent and quality. It also lack the implementation of some Web technologies, like Java applets, JavaScript, application of XSLT stylesheets, that today are a fundamental component of every real-world browsing application.

Mozilla has recently enabled MathML support by default and version 1.0 is about to be released. Mozilla is a complex and somewhat cumbersome application whose core engine,

[6]A comprehensive list of these applications can be found at `http://www.w3.org/Math/implementations.html`

called Gecko, can be used independently. In fact, there are browsers[7] that are built around Gecko and Netscape uses Mozilla's source code for its Netscape Navigator Web browser.

This characteristic is relevant since support for MathML inside Gecko automatically implies support for MathML in every application that is based around it, although Netscape has not decided a clear policy with respect to MathML yet.

7.2.2 Extensions

By extensions we mean components, such as Java applets or ActiveX controls, that work together with some hosting application, typically a Web browser, enabling the rendering of MathML markup embedded in HTML.

The problem of these external components is that they do not integrate seamlessly with the rest of the document (in many cases they render as inert pictures). That means that uniform interaction (selection, printing, editing) with the rest of the document is compromised.

The use of such extensions is also platform dependent, and can hinder the diffusion of markup-languages as each extension usually has a peculiar mechanism to process the markup to be rendered.

7.2.3 Indirect Rendering

Indirect rendering is the conversion of MathML markup, by means of an XSLT stylesheet or a dedicated processor, to a different format (TEX or SVG) for which a suitable rendering engine is available. Examples of such tools are the MathML to SVG converter by Schemasoft (`http://www.schemasoft.com/MathML/`) or the MathML to TEX translator developed at the Ontario Research Centre for Computer Algebra (`http://www.orcca.on.ca/MathML/`).

In both cases these attempts mainly derive from the lack of high-quality rendering engines for MathML. As such, they can be suitable in the short term, as long as there are not many implementations available, but we believe that such approaches should be eventually abandoned.

A particularly severe drawback of the conversion from MathML to SVG is the verbosity of the resulting SVG markup. This is even more true in those cases where the single characters in MathML markup are eventually rendered in SVG using spline curves (and not some native fonts).

7.3 Requirements

An adequate rendering system for MoWGLI must take into account the following aspects:

1. high level of compatibility with the overall architecture, which is based on XML as the central technology for storing, retrieving and processing mathematical documents;

2. Web-friendliness, intended as the ability of exploiting commonly available (not specialised) Web services and applications for simple operations (like browsing) in the repository of information;

3. presentation seen as a vehicle for semantic information, by means of direct inclusion of, or indirect reference to, content-oriented information corresponding to the rendered document (or part of it);

[7]Galeon, see `http://galeon.sourceforge.net`

4. independence from the output device used for rendering, but also adaptability to different environments.

As MoWGLI has a strong commitment to XML technologies and focuses on mathematical documents, MathML presentation markup, used in conjunction with HTML or Formatting Objects, is the most natural and appealing candidate meeting these points.

Nevertheless, currently available implementations of MathML are certainly not sufficient. While current browsing applications can serve the needs of casual users of the library very well, much of the emphasis of the project is on specialised services for searching operations, retrieval, assisted authoring and editing of mathematical documents. These activities all require the development of specific interfaces and applications where rendering and interaction enable access to the underlying information.

The implementation of a standalone rendering engine for MathML will allow us to develop specific applications in the context of MoWGLI, and to target the engine to the requirements of the project.

At this early stage of the project, we can summarise what appear to be important characteristics of such standalone engine:

- fairly complete and robust implementation capable of rendering MathML markup;
- easily embeddable in applications and tools;
- easily usable from different programming languages, as developers of the project will prefer to work with languages that are suitable to the task to be addressed;
- possibly usable as an extension to current browsing applications, so to enrich the set of such tools;
- modular and "open", so to ease its integration with other components designed along the same lines.

The last point, in particular, suggests that the engine may serve as the starting point for the development of a modular, component-oriented rendering engine for XML-based markup languages. This kind of engine could fill the gap between "fat", monolithic applications (like Mozilla), and browsers using plug-ins that hinder and constrain integration of foreign markup within larger documents.

8 Interaction, Editing and Other Dynamic Aspects

Among the great benefits we expect from on-line repositories of mathematics, there is the possibility of interacting with the so-far static mathematical content. In particular, the more complex possible form of interaction is WYSIWYG or almost WYSIWYG editing, that we will address in section 8.6. We now describe the different forms of interaction ordered by increasing complexity.

8.1 Hypertexts

The easiest form of interaction is the possibility of following an hyperlink. For us, one important class of links are the links between every occurrence of a concept and its definition. We believe that adding such a fine-grained hypertext structure to mathematical documents can really help the reader to understand the content, especially when the reader does not remember every definition previously introduced in the document or in related documents. In order to avoid burdening the writer with the insertion of an hyperlink for every occurrence, we must require some kind of automatization during the editing. Actually no on-line distributed mathematical document provides so fine-grained hyperlinks. In fact, PostScript does not allow hyperlinks at all, XHTML+MathML documents are not yet diffused and we know of no example of PDF document with that characteristic.

The XML standard technologies to provide links to documents are XLink [38] and XPointer [44], which can be orthogonally composed with any other XML dialect, in particular MathML and XHTML. XLink provides a quite complex model for hypertext, comprising multi-directional links and links between more than two resources. In particular, multi-directional links provide an added value to simple links, allowing the possibility to require for each concept its list of occurrences. XPointer, instead, is an extension of XPath to precisely identify the source and targets of an hyperlink without having to alter the source and target document markup. As a consequence, it becomes possible to link any point of a third-party document.

All the current widespread browsers already support at least simple XLinks, which are uni-directional single-target links "à la HTML". Moreover, there are some solutions [35] that does not require changes to the browser to extend the implementations to the full XLink technology. XPointer, instead, is still basically unsupported.

Independently from the available technologies, there is at least another problems to be solved before fine-grained mathematical hypertext can become a reality: especially in the realm of formal mathematics we do not usually want to look at all the details of a proof, especially when we are able to understand a development even without them. How should we deal with this hidden information when trying to follow an hyperlink?

Let's examine a small scenario in the realm of implicit coercions, which are conversions between data-types which are automatically applied by the system and usually hidden to the user: *We have defined both a property on semigroups and an implicit coercion from a ring to its underling multiplicative semigroup. A user finds an occurrence of the property applied to a ring, and does not know to which of the two underlying semigroups the property is being applied. So, he wants to click somewhere to get the definition of the implicit coercion, but no markup for the implicit coercion is showed!* A solution is to give the user the possibility to ask the browser to show also the coercion. This solution is too naif, because the user can be confused enough to not suspect the existence of the coercion. Thus, when developing our interfaces to the library, we must address this problem very carefully.

8.2 Dependency Graphs

Fine grained hypertexts are really useful when trying to understand a single formula or theorem that is based on something whose definition we do not remember. They do not help much when we have a big development and we want to understand the global relationship within the objects.

A user is willing to study the fundamental theorem of algebra, but he realizes that he does know nothing about ordered fields. Which part of the development will he be unable to understand? Which other parts must he study before being able to understand ordered fields? Which parts can he skip?

To answer the above questions, we need two sources of information. The first one is just meta-information describing the dependencies between parts of the library. This information is already implicitly stored in the hypertext as the transitive closure of the relation between an occurrence and its definition. The second one is the user itself, who must provide the description of his own knowledge.

The net of hyperlinks forms a directed graph of dependencies and the rendering of this graph provides a useful level of understanding of the global structure of a development. This approach has been already successfully applied in different situations, as the analysis of the dependencies between Linux kernel sources[8], debian packages[9] and Ocaml modules[10].

Some tools already exist for the layouting of directed graphs [24], and SVG [42] is the XML-based standard language for describing vector graphics. Moreover, SVG is so far one of the XML recommendation with the highest number of implementations, it is natively supported on the most common browsers and it should interact well with other XML standards as MathML and XLink [43]. So, the technology for graph rendering is mature enough and we should face no technological problem here. Nevertheless:

- Especially in the context of mathematics, non-trivial theorems recursively depend on hundreds of other concepts. So we can not render the whole dependency graph. In any case, a huge graph will not be informative.
- In particular, the displaying of the part of the graph that describes trivial developments that are already clear to the user can be especially annoying. More generally, the user may be interested only in those parts of a development whose structure he has not understood yet.
- Every user has a different profile, i.e. he knows different things and he considers trivial different parts of math.
- Some concepts are so closely related that they should probably be collapsed in a single node of the tree.

The above considerations imply the need to to generate the graphs accordingly to the user profile. We should also give to the user the possibility of browsing the graph, dynamically pruning the branches he is not interested in and zooming on the interesting parts.

The kinds of interactions above are not as trivial as following an hyperlink, because they require the possibility of generating a different document depending on the user input (its

[8]http://lwn.net/2002/0103/a/hviz.php3

[9]http://bobo.fuw.edu.pl/cgi-bin/man2html/usr/share/man/man8/apt-cache.8.gz

[10]http://www.research.att.com/ trevor/ocamldot/

profile) and the possibility of interactively modify an already generated document. In the next sections we address both these kinds of problems in a more general setting.

8.3 Parametric Documents

A parametric document is a document whose generation may depend on a set of parameters. Once generated, the document will not be changed by user interactions, unless a new generation is triggered again. The most part of the documents we are interested in are parametric. For example, they may depend on

- The required rendering format.
- The browser capabilities.
- The required mathematical notation. Mathematical tradition has developed different notations for the very same concept and we must allow the reader to choose his preferred one.
- The level of details required. This is a critical point especially in the realm of formal mathematics.
- The user knowledge and aims.
- The set of mathematical repositories the user want to have access to.
- The set of service providers the user want to have access to.

Let us define a user-profile as a set of preferences. The main issue we must face is where and how the user-profile must be stored, and how can it be changed. In particular we must face some conflicting requirements:

- The user should be able to store its profile locally, so that he can access the library even if he is off-line.
- The user should be able to store its profile on a well-known server, so that he can use his settings independently from the host he is using.
- We must have an intuitive interface to set the user-profile and reduce the user-configurable options as much as possible.
- The system should give the user as much control as possible on the document rendering.

We must also face the problem that the HTTP protocol is basically connectionless. So, if the user-profile is stored on the client host, it must be sent to the server every time a page is requested. Moreover, at least part of the user-profile must be sent at least once per session to the server (e.g. the browser capabilities). Storing the user profile on the server may be a too centralised solution, especially in the case of multiple service providers that need the user-profile and that are possibly running on different hosts widespread on the net.

Note that in the previous discussion we have assumed that it is not feasible to generate a parametric document on the server and instantiate it on the client. In fact, the browser technology is so quickly evolving and great efforts have been made towards increased standardisation that we can more and more hope to leave the instantiation to the client. So, during the development of MoWGLI we will have to monitor this possibility.

One project that is firmly based on parametric mathematical documents is ActiveMath. The ActiveMath learning environment achieves the generation of content "books" according to a user-model and to the dependencies of OMDoc items. The original role of the ActiveMath server is that of a learning environment.

It presents OMDoc content on the Web, features an updating user-model, a course generator, a suggestion mechanism, and interactive exercises connected to mathematical systems.

It is extensively described in [18] and its usage of the knowledge representation is described in [31].

One of the mechanism to assemble old content in new different ways, i.e. to provide a new view on the library, relies on table-of-contents which are lists of pointers to content items. These tables-of-contents can, of course, recreate an existing sequence of OMDoc items. One of the first re-usability, however, is the ability for a teacher or journal editor to create different books using new table-of-contents. New books can also be created by the course generator, a system that chooses the content to be presented following dependencies between items (encoded in the metadata) and that applies *pedagogical rules* to choose the necessary items (according to the user-model) and order them in an appropriate fashion.

The user model is updated with such information as the time the user spends on reading each item and the performance result on exercises. Though probably less applicable in a mathematical research practice, this ability is important as the assessment of the user knowledge is a tedious and error prone mechanism.

8.4 Semantic Selection

In modern graphical user interfaces, the basic operation to implement complex interactions is selection. In MoWGLI most parts of the documents have an underlying content or semantic encoding, and we expect to provide interactions with the underlying content. Thus, we need a different basic operation, which is *semantic selection.* Semantic selection means being able to select a node of the DOM of the content-level or semantic-level document, highlighting its rendering. The reason for this requirement is that we are often interested in operations that are meaningful only at the content or semantic level. Some examples are:

- Doing cut&paste towards a computer algebra system, whose input is MathML content and not MathML presentation.
- Asking for an expression to be simplified.
- Collapsing the parts of a proof we are not interested in.

Semantical selection is a complex task. First of all it requires some machinery to retrieve the content node that generated a presentation node. One possible solution is to use XLink to store in the presentation file links to their content sources. Because the mapping from content to presentation is usually performed by XSLT stylesheets, the latest are responsible of generating all the links required. Once XLinks are available, we must still constraint the user to select only a region that corresponds to the rendering of some content node. Note that, in the general case, a region is not required to be contiguous. Thus selection can not be performed by the browser alone, but we need a user-defined control logic that must listen for selection events generated by the browser, choose the candidate region to be highlighted (for example as the smallest enclosing region which is the image of a content node) and instruct the browser to select that region. More complex forms of selection could be also useful. For example, one natural extension is the possibility to select a set of content or semantic nodes.

It may be possible that semantical selection turns out to be not implementable in a reliable way in some browsers. Moreover, once a node is selected, we must still apply the wanted transformation at the content or semantics level and map it to the corresponding transformation on the DOM tree. This usually require some interaction with an external service provider that may be, for example, a computer algebra system or a proof-assistant. This interaction may be issued by a client-side script or may require something more complex, as a Java applet. In this case we may consider the possibility of encoding the whole selection logic inside the applet.

8.5 Dynamic Documents

A Dynamic Document is a document on which we allow interactions that change the document content without need of generating it again from scratch. The change can be triggered by the user, by a timer that expires or by another program. In the realm of XML, this corresponds to alter the DOM [45] tree of the document, triggering DOM events [46] that propagates from the affected node towards the root of the tree. Listeners can request to be notified when some events are fired. EcmaScript [29] bindings to the DOM interface are usually available on the last generation browsers such as Internet Explorer 6 and Mozilla. The available Plug-ins and ActiveX to render MathML, instead, do not expose the DOM interface of their document. As a consequence, dynamic documents with MathML parts rendered by means of ActiveX or Plug-ins are quite hard to develop and become absolutely not-portable. Considering that under Internet Explorer there is no native MathML support, this may be a big limitation to the development of active mathematics on the web.

Sometimes the set of instances a dynamic document may have is finite. For example, if we are showing a proof, we can expect each step of the proof to be shown or to be hidden and we want to give the possibility to the user to dynamically change the set of hidden steps. In this case we can simply generate a document in which we record all the different renderings of the same node, and use a script to control the set of renderings that is currently used. In all the other cases, we must be able to change only a subset of the content or semantic DOM and we have to apply the transformations to the new tree to get the parts of the presentation DOM that must be replaced. When the transformations may be performed only locally, we can expect the rendering time of the modified fragment only to be sensitively smaller than the rendering time of the whole document.

8.6 WYSIWYG Editing

Probably the most complex form of interaction is WYSIWYG editing. In the WYSIWYG (What You See Is What You Get) approach, editing is performed either by inserting new text in a certain position or by selecting part of the document and performing some action on it.

WYSIWYG editors of mathematical formulas usually are structured editors, where a formula is entered by selecting a hole in it and filling it with a new operator applied to new holes. The operator to be entered is usually chosen from a palette of operators. This is, for example, the approach of the MS Word Equation Editor and of its professional edition, that is the MathType Equation Editor[11].

[11]Design Science MathType, http://www.dessci.com/company/press/releases/oct01.stm

Even if, according to the technical annex of the MoWGLI contract, WYSIWYG editing is outside the scope of the project, it could be worth implementing some basic support for it. For example, a user may want to search the library for a particular formula and he should be provided with a user-friendly interface to enter it. The Java OpenMath Editor[12], developed at UNSA (University of Nice Sophia Antipolis), is such a tool and it's usability (and its extension to support dynamic enrichment of symbols) should be evaluated along with several other projects aiming at the visual edition of semantic mathematical content.

The activity of inserting mathematical formulas in a MoWGLI document should always privilege forms of input based on content representation, having automatic or semi-automatic tools to produce presentation data from the content data. The main novel difficulty of content WYSIWYG editing with respect to traditional presentation WYSIWYG editing lies in semantic selection, as already explained in section 8.4. Edition of semantic markup may, however, offer more support to the user. The ability to provide types to the parts of the mathematical object being edited can offer a strong support, for example, in the choice of the symbols that have to be inserted.

8.6.1 Alternatives to WYSIWYG Editing

The WYSIWYG approach, which is beginner-friendly, is sometimes too cumbersome for experienced users that usually prefer a non-WYSIWYG editor as TeX or a semi-WYSIWYG solution where formulas are quickly typed in a textual encoding and they are then rendered on-the-fly[13]. Moreover, most of the visual editors and input syntax tend to be non-extensible (this is the case of MathType Equation Editor) and they usually don't support semantics (LyX, TeX, ...).

Domain specific parser, that use concise text for manual input and produce XML content data from this input, may be a valuable alternative. An example of such tool is the QMath parser[14]: its line-based input syntax creates OMDoc documents with OpenMath objects for formula objects. The QMath syntax for formulas is somewhat close to the Computer Algebra Systems. Operators and their precedences and function applications are allowed; moreover, the ability to input any character of the Unicode range allows the use of much more symbols than what LaTeX allows. The input is compact and readable, and can be extended at any time.

Experience has shown that purely automatic tools to generate presentation data from content data can sometimes make wrong choices, leading to badly typeset formulas. Means to let the user guide the typesetting algorithms need to be investigated. One should avoid a situation (as in LaTeX) where the user indicates the precise location of linebreaks, rather choosing between quick and naive typesetting or clever but complex typesetting.

[12]The Java OpenMath Editor, JOME, is slightly documented in `http://mainline.essi.fr/jome/jome-en.html`

[13]LyX, http://www.lyx.org/

[14]The QMath processor is documented at `http://www.matracas.org/`

References

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

[2] Autexier, S., Hutter, D., Mantel, H., and Schairer, A., “Towards an evolutionary formal software-development using CASL”, in Choppy, C., and Bert, D., eds., *Proceedings Workshop on Algebraic Development Techniques, WADT-99*, LNCS 1827. Springer, (2000).

[3] Autexier, S., Hutter, D., Mantel, H., and Schairer, A., “OMDoc: Towards an internet standard for the administration, distribution and teaching of mathematical knowledge”, in Roanes Lozano, E., ed., *Proceedings of ”Artificial Intelligence and Symbolic Computation”*, LNAI 1930. Springer, (2001).

[4] Borwein, P., Times Higher Education Supplement 1514, 23 November 2001.

[5] Caprotti, O., and Cohen, A.M., “Draft of the Open Math standard. The Open Math Society”, (1998), `http://www.nag.co.uk/projects/OpenMath/omstd/`.

[6] Caprotti, O., Geuvers, H., and Oostdijk, M., “Certified and Portable Mathematical Documents from Formal Context”, Proceedings of the First International Workshop on Mathematical Knowledge Management, September 2001.

[7] Language Design Task Group CoFI, *Casl — the CoFI algebraic specification language — summary, version 1.0.*, (1998). For a related online version see: `http://www.brics.dk/Projects/CoFI`.

[8] Coscoy, Y., *Explication textuelle de preuves pour le Calcul des Constructions Inductives*, PhD thesis, (Université de Nice-Sophia Antipolis, 2000).

[9] Coscoy, Y., Kahn, G., and Thery, L., *Extracting Text from Proofs*, (INRIA Sophia Antipolis), Report no. RR-2459.

[10] Amerkad A., Bertot Y., Rideau L. and Pottier L., “Mathematics and proof presentation in Pcoq”, Proceedings of the Workshop Proof Transformation and Presentation and Proof Complexities (PTP’01), June 2001, Siena, Italy.

[11] Filliatre, J.-C., “CoqDoc: A Documentation Tool for Coq”, `http://www.lri.fr/~filliatr/coqdoc/`.

[12] Hutter, D., *Reasoning about theories*, (Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1999).

[13] Knuth, D.E., *The Tex book*, (Addison-Wesley, Reading, MA, USA, 1998).

[14] Kohlhase, M., “OMDoc: An Infrastructure for OpenMath Content Dictionary Information”, Bulletin of the ACM Special Interest Group for Algorithmic Mathematics SIGSAM, (2000).

[15] Kohlhase, M., *OmDoc: An open markup format for mathematical documents*, (Fachbereich Informatik, Universität des Saarlandes, 2000), Report no. SR-00-02. For a related online version see: Mathweb, `http://www.mathweb.org/omdoc`.

[16] Libbrecht, P., Büdenbender, J., Frischauf, A., Goguadze, G., and Ullrich, C., "Presentation and Re-Use of XML-content in ActiveMath", unpublished.

[17] Loeckx, J., Ehrig, H.-D., and Wolf, M., *Specification of Abstract Data Types*, (Teubner, Chichester, New York, Brisbane, 1996). ISBN: 3-519-02115-3.

[18] Melis, E., Büdenbender, J., Andres, E., Frischauf, A., Goguadze, G., Libbrecht, P., Pollet, M., and Ullrich, C., "Activemath: A generic and adaptive web-based learning environment", *Artificial Intelligence and Education*, **12**(4), (2001).

[19] Naylor, B., and Watt, S., "Meta Style Sheets for the Conversion of Mathematical Documents into Multiple Form", On-line Proceedings of the First International Workshop on Mathematical Knowledge Management, (2001).

[20] Nederpelt, R., and Kamareddine, F., "An abstract syntax for a formal language of mathematics", Proceedings of the Fourth International Tbilisi Symposium on Language, Logic and Computation, September 2001, Borjomi, Georgia.

[21] Odlyzko, A.M., "Tragic Loss or good riddance? The impending demise of traditional scholarly Journals", *Intern. Journal of Human-Computer Studies*, **42**, 71–122, (1995).

[22] Oostdijk, M., *Generation and Presentation of Formal Mathematical Documents*, PhD thesis, (University of Eindhoven).

[23] Russ, S.R., Kechris, A.S., Pillay, A., and Shore, R.A., "The prospects of mathematical logic in the twenty-first century", *Bulletin of Symbolic Logic*, **7**(2), (June, 2001). Studies 42 (1995), 71–122.

[24] AT&T Labs-Research, "The Graphviz Suite", `http://www.research.att.com/sw/tools/graphviz/download.html`.

[25] "Common Criteria for Information Technology Security Evaluation. Part 1: Introduction and General Model, Version 2.1.", (August, 1999), `http://csrc.nist.gov/cc/`.

[26] "Common Criteria for Information Technology Security Evaluation. Part 2: Security Functional Requirements, Version 2.1.", (August, 1999), `http://csrc.nist.gov/cc/`.

[27] "Common Criteria for Information Technology Security Evaluation.Part 3: Security Assurance Requirements, Version 2.1.", (August, 1999), `http://csrc.nist.gov/cc/`.

[28] "The Coq Proof Assistant Reference Manual, Version 7.2.", INRIA Technical Report 255, (2002).

[29] "ECMAScript Language Specification, Standard ECMA-262", `http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM`.

[30] "The JavaDoc Tool Homepage", `http://java.sun.com/j2se/javadoc/`.

[31] "Knowledge representation and management in ActiveMath", The Active Math Group. Submitted to the Proceedings of MKM'01 in Annals of Mathematics and Artificial Intelligence, (2002).

[32] "Living Reviews in Relativity", `http://www.livingreviews.org`.

[33] *PostScript Language Reference Manual, Third Edition*, (Addison-Wesley, 1999). For a related online version see: `http://partners.adobe.com/asn/developer/pdfs/tn/PLRM.pdf`.

[34] *PDF Reference: Adobe Portable Document Format", Version 1.3*, (Addison-Wesley, 2000). For a related online version see: `http://partners.adobe.com/asn/developer/acrosdk/DOCS/PDFRef.pdf`.

[35] University of Bologna, "The XLinkProxy", `http://www.cs.unibo.it/~fabio/XPointer/`.

[36] W3C Consortium, "Extensible Markup Language (XML) 1.0 (Second Edition)", `http://www.w3c.org/TR/2000/REC-xml-20001006`. W3C Recommendation 6 October 2000.

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

[38] W3C Consortium,"XML Linking Language (XLink) Version 1.0", `http://www.w3.org/TR/2001/REC-xlink-20010627/`. W3C Recommendation 27 June 2001.

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

[40] W3C Consortium, "HTML 4.01 Specification", `http://www.w3.org/TR/html4/`. W3C Recommendation December 24, 1999.

[41] W3C Consortium, "Extensible Stylesheet Language (XSL), Version 1.0", `http://www.w3.org/TR/xsl/`. W3C Recommendation, 15 October 2001.

[42] W3C Consortium, "Scalable Vector Graphics (SVG) 1.0 Specification", `http://www.w3.org/TR/SVG/`. W3C Recommendation, 4 September 2001.

[43] W3C Consortium, "An XHTML + MathML + SVG Profile", `http://www.w3.org/TR/2002/WD-XHTMLplusMathMLplusSVG-20020430/`. W3C Working Draft 30 April 2002.

[44] W3C Consortium, "XML Pointer Language (XPointer) Version 1.0", `http://www.w3.org/TR/2001/CR-xptr-20010911/`. W3C Candidate Recommendation 11 September 2001.

[45] W3C Consortium, "Document Object Model (DOM)", `http://www.w3c.org/DOM/`.

[46] W3C Consortium, "Document Object Model (DOM) Level 2 Events Specification, Version 1.0", `http://www.w3.org/TR/DOM-Level-2-Events/`. W3C Recommendation, 13 November 2000.