Towards a Library of Formal Mathematics

Andrea Asperti, Luca Padovani, Claudio Sacerdoti Coen, Irene Schena
Department of Computer Science
Via di Mura Anteo Zamboni 7, 40127 Bologna, ITALY.
contact: asperti@cs.unibo.it

Abstract

The eXtensible Markup Language (XML) opens the possibility to start anew, on a solid technological ground, the ambitious goal of developing a suitable technology for the creation and maintenance of a virtual, distributed, hypertextual library of formal mathematical knowledge. In particular, XML provides a central technology for storing, retrieving and processing mathematical documents, comprising sophisticated web-publishing mechanisms (stylesheets) covering notational and stylistic issues. In this paper, we discuss the overall architectural design of the new systems, and our progress in this direction (http://www.cs.unibo.it/~asperti/HELM/home.html).

1 Introduction

Existing logical systems are not suitable for the creation of large repositories of structured mathematical knowledge accessible via Web. In fact, 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 clearly 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 with its new emphasis on *content*. Moreover, the information provided by such libraries usually lacks of a satisfactory form of presentation. This is a separate, still parallel aspect which is undoubtedly fundamental to achieve a significant dissemination of mathematical knowledge.

The eXtensible Markup Language (XML, see [2]), whose aim is to encode information according to its structure and content, is rapidly imposing as the main tool for representation, manipulation, linking and exchange of structured information in the networked age. In this paper we advocate the pivotal role of XML in the development of a suitable technology for the creation and maintenance of large repositories of structured mathematical knowledge, and describe the overall architectural design of the new systems.

The feasibility of describing mathematical structures using a markup language is already testified by the MathML project[1] [3]. The Mathematical Markup Language is an instance of XML for describing mathematical expressions capturing both their presentation and content. Although the emphasis of MathML is just on mathematical expressions, (while we are also concerned with different mathematical entities such as proofs, definitions, theorems, sections, theories, metadata and so on) MathML is an essential component of our architecture, as discussed in section 6 (see also [8]).

Let us finally remark that the broad goal of the project goes far beyond the trivial suggestion to adopt XML as a neutral specification language for the "compiled" versions

[1] We joined the MathML Working Group of the World Wide Web Consortium in October 1999.

of the libraries, or even the observation that in this way we could take advantage of a lot of functionalities on XML-documents already offered by standard commercial tools. First of all, having a common, application independent, meta-language for mathematical proofs, similar software tools could be applied to different logical dialects, regardless of their concrete nature. This would be especially relevant for all those operations like searching, retrieving, displaying or authoring (just to mention a few of them) that are largely independent from the specific logical system. Moreover, 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. Finally, this "standardization" process naturally leads to a substantial simplification and re-organization of the current, "monolithic" architecture of logical frameworks. All the 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.

Readers already acquainted with XML can skip section 2, where we give a brief introduction to the language, and start directly with the general overview (section 3).

2 The eXtensible Markup Language

Perhaps, the best way to introduce XML in few lines is to take a look at a simple example, given below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<recipebook>
  <recipe>
    <recipetype name="sweets"/>
    <title>Apple Cake</title>
    <ingredient>apples</ingredient>
    <ingredient>meal</ingredient>
    <ingredient>sugar</ingredient>
    <step number="1">
       ...
    </step>
    <step number="2">
       ...
    </step>
  </recipe>
</recipebook>
```

XML gives a method for putting structured data in a text file. Roughly speaking, the XML specification says that a XML document is made of *tags* (words bracketed by '<' and '>'), *attributes* (of the form `name="value"`) and text. Tags are used to delimit *elements*. Elements may appear in one of the following two forms: either they are non-empty elements, as `recipe` or `ingredient` (they can contain other elements or text), or they are empty elements, as `recipetype`.

The XML specification defines a XML document to be well-formed if it meets some syntactical constraints over the use of tags and attributes. For example, non-empty elements must be perfectly balanced. For this reason, someone can think of tags of non-empty elements as labelled brackets for structuring the document.

It is remarkable that XML does not specify any predefined tag set at all. Rather, it lets the user specify his own grammar by means of a Document Type Definition (DTD), a document which defines the allowed tags, the related attributes and which is the legal content for each element. The XML specification just defines the validity of a XML document with

respect to a given DTD. This is why XML is a *meta-language* that can be instantiated to a potentially infinite set of languages, each with its own DTD.

For example, the document above is valid with respect to the following DTD:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT recipebook (recipe)+ >
<!ELEMENT recipe (recipetype, title, ingredient+, step+) >
<!ELEMENT title (#PCDATA) >
<!ELEMENT ingredient (#PCDATA) >
<!ELEMENT step (#PCDATA) >
<!ATTLIST step number CDATA #REQUIRED >
<!ELEMENT recipetype EMPTY >
<!ATTLIST recipetype name CDATA #REQUIRED >
```

We can note that a `recipebook` tag can contain one or more `recipe` elements, and that each `recipe` can contain exactly one `recipetype` followed by exactly one `title`, a positive number of `ingredient`s and at least one `step` element. `title` is an example of element containing text only (`PCDATA`). `name` and `number` are attributes of `step` and `recipetype` tags respectively. The keyword `REQUIRED` states that an attribute is mandatory, i.e. it cannot be omitted when using its related tag.

References to Documents Documents and resources in general must have a name in order to be accessible over the Web. This is accomplished via the use of URIs (Universal Resource Identifiers) as defined in [10]. A generic URI is made of a formatted (structured) string of characters, without any intended meaning. URLs (Uniform Resource Locators) are a particular kind of URI specifically designed to name resources accessed by means of a given protocol (for example, HTML documents are accessed via the HTTP protocol).

As an example, let us suppose that the DTD above is stored in a document whose URI is `bookstore:/books/recipebook.dtd`; we can associate this DTD to a XML recipe book adding a special prologue:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE recipebook SYSTEM "bookstore:/books/recipebook.dtd"!>
<recipebook>
...
</recipebook>
```

Since URIs are designed to be arbitrarily extensible, standard browsers and processing tools can only be required to handle URLs, while URIs are meant to be processed by specific applications aware of them.

3 The HELM project

The overall architecture of our project, the Hypertextual Electronic Library of Mathematics, is fully described in Figure 1.

Once XML has been chosen as the standard encoding format[2] for the library, we must face the problem of recovering the already codified mathematical knowledge. Hence, we need new modules implementing exporting functionalities toward the XML representation for all the available tools for proof reasoning. Currently, we have just written such a module only for the Coq proof assistant [9]. In the near future, we expect that similar exporting functionalities

[2] A standard *format*, not a standard *language*!. In other words, the standardisation we are pursuing is not at the *logical* level, but at the *technological* one.

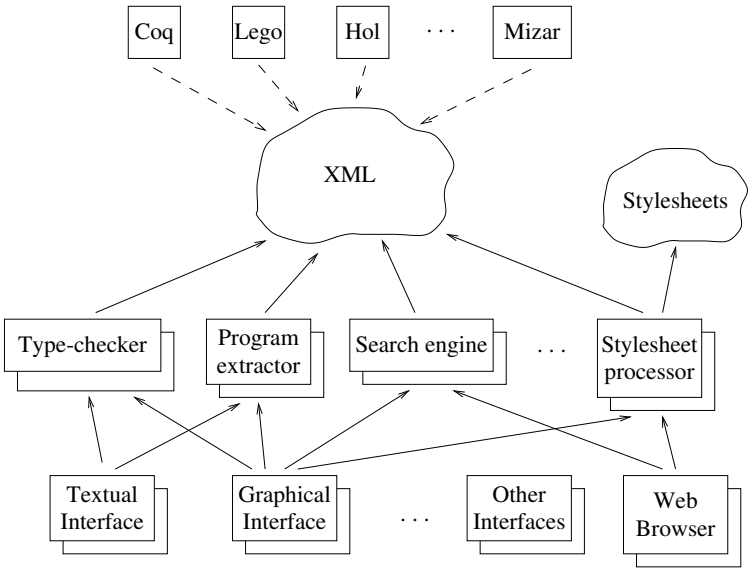

Figure 1: Architecture of the HELM project.

will be provided by the developers of the other logical systems. We will describe the exporting issues in section 4.

To exploit and augment the library, we need several tools to provide all the functionalities given by the current "monolithic" proof assistants, such as type-checking, proof searching, program verification, code extraction and so on. Moreover, we can use the available well-developed and extensible tools for processing, retrieval and rendering of XML-encoded information. In particular, to render the library information, we advocate the use of stylesheets, which are a standard way of associating a presentation to the content of XML files. This allows the user to introduce new mathematical notations by simply writing a new stylesheet. In section 5 we shall briefly discuss our implementation of a type-checking tool, while in section 6 stylesheets are addressed in details.

The user will interact with the library through several interfaces that integrate the different tools to provide an homogeneous view of the functionalities. We are developing two interfaces, described in section 7.

Because of the nature of the library, we have also provided a model of distribution, which is discussed in section 8.

4 Exporting from Coq

Coq [9] is one of the most developed proof-assistant, based on a very rich logical framework called the Calculus of (Co)Inductive Constructions (CIC). The great number of functionalities (proof editing, proof checking, proof searching, proof extraction, program verification) have made the system very big and complex. In order to work on the information encoded in such a system, the only practical way is that of writing a new module that extends it, gaining access to its internal representation.

Finding the right information inside the system itself is not a trivial task: first of all, information is encoded inside the data structures of Coq and data structures change from one version of the system to another; secondly, the information searched is often not directly available. For example, when a file is loaded, its path is completely forgotten, even if this

information could be necessary thereafter (e.g. we need it for exporting). Due to these difficulties, Coq has proven itself a challenging test bench for exporting information to XML.

To do the exporting, we have written a module in which we have implemented a set of top-level commands that, given the name of one or more CIC objects (variables, constants, types or axiom definitions), generate the corresponding XML file. The choice of writing a module without modifying the system itself seemed to be the best one, but we had to cope with the problem of information not directly available at this level. For example, the fact that the file pathnames are not present has forced us to structure the exported files into directories during a second phase. In this further phase, when moving files, we have also to modify all the URIs in all the files to reflect the changes of their position. This solution and those to similar problems are not acceptable because they add too much complexity without being necessary: we hope that the exporting functionality will be integrated within the system itself.

To design the module, the first difficulty has been the identification of which information should be exported and what should be its structure. We have chosen not to export:

Parsing and pretty-printing rules Parsing rules should depend only on the proof engine. To be able to use other proof engines different from Coq we need not to rely on Coq's own rules. Similarly, pretty-printing rules should depend only on the users choice and the type of available browser.

Tactics-related information These, too, are proof engine dependent. Moreover, we do not think that the tactics used to do a proof are really meaningful to understand the proof itself (surely, they are not the real informative content). In fact, the level of tactics and the level at which a proof should be understood are not the same: what some simple tactics do (as "Auto" that automatically search a proof) is not at all obvious. Moreover, the sequence of tactics used is clearly reflected in the lambda-term of the proof; hence it is possible to add as an attribute to a subterm the name and parameters of the tactic used to generate it.

Redundant information added by Coq to the terms of CIC Coq adds in several places a lot of information to CIC terms in order to speed up the type-checking. For example, during the type-checking of an inductive definition, Coq records which are the recursive parameters of its inductive constructors; this information is then used during the type-checking of fix functions to ensure their termination. This is an example of a clearly redundant information, useless for browsing purposes and that could be useless also for other type-checkers. We have then decided to discard it accordingly to a **principle of minimalism**: *no redundant information should be exported.* If the principle were not followed, every time we use an XML file we would have to add checks to verify the consistency of the redundant information.

Sometimes Coq also adds some non-redundant rendering information, for example when the user asks the system to infer a type and does not want to view the inferred type thereafter. This information will eventually be exported, even if this is not implemented yet.

The remaining, interesting information could be structured into three different levels that we have clearly separated in the XML representation. The first one is the *level of terms.* Terms (or expressions) could never appear alone, but only as part of an object definition. In Coq the terms are CIC lambda-expressions, i.e. variables (encoded as DeBrujin indexes), lambda-abstractions and applications, product types and sorts, augmented with a system of inductive types in the spirit of the ones of Martin-Löf, comprising (co)inductive types and constructors, case analysis operators and inductive and coinductive function definitions. The whole level is extremely dependent from the logical framework.

The second level, that uses the previous one to encode both bodies and types, is the one of *objects*. Every object is put into a different file. The files are structured into directories that corresponds to sections in Coq, i.e. delimiters of the scope of a variable. Sections are also used in Coq to structure a large theory into subtheories. In HELM, the former usage is retained, while theories are described in another way (see the third level).

Constants (definitions/theorems/axioms) Constant objects of Coq are used to represent definitions, theorems and also axioms. Definitions and theorems are syntactically identical and have a body and a type. The only difference is semantical: theorems are usually opaque (only their type is used in CIC terms) because of proof-irrelevance, while definitions are transparent (their body is substituted in their occurrences in other terms during type-checking). Axioms, instead, are constants with a type but without a body. We choose, during extraction, to discriminate only axioms from definitions and theorems, that become indistinguishable once extracted. We leave to the next level (that of theories) the responsibility of "marking" non-axiom constants as theorems, definitions, lemmas, facts, ...

Variables Variables have only a type and not a body. A variable behaves like an axiom inside the section where it is defined and as a parameter when referring to another object of that section. Hence, sections are used to delimit the scope of variables.

(Co)Inductive Definitions In Coq, blocks of mutual (co)inductive definitions can also be defined. Each definition inside such a block has a name, a type (called arity in Coq) and a possibly empty list of constructors. Each constructor has a name and a type. A simple example is the inductive type of natural numbers whose name is nat, whose type is Set and whose constructors are O of type nat and S of type (nat $\rightarrow$ nat).

Blocks, as constants, could depend on variables; the list of variables (parameters) on which all the definitions in the block depend is also exported from Coq.

Proof in progress We choose also to export unterminated proofs. As terminated theorems, an unterminated proof has a name, a type and a body; moreover, it has also a list of conjectures on which the body depends. Each conjecture has a type but not a body: to end the proof you must provide a body for each conjecture.

An example of an object file describing a constant (a theorem) can be found in appendix A. Another one is the following where you can see the definition of the inductive type of natural numbers:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE InductiveDefinition SYSTEM "http://localhost:8081/getdtd?url=cic.dtd">
<InductiveDefinition noParams="0" params="">
 <InductiveType name="nat" inductive="true">
  <arity><SORT value="Set"/></arity>
   <Constructor name="O">
    <REL value="1" binder="nat"/>
   </Constructor>
  <Constructor name="S">
   <PROD>
    <source><REL value="1" binder="nat"/>
    </source>
    <target><REL value="2" binder="nat"/>
    </target>
   </PROD>
  </Constructor>
 </InductiveType>
</InductiveDefinition>
```

Tags of the term level (`PROD`, `REL`, `SORT`) have all the letters capitalized. Tags of the object level (`InductiveDefinition`, `InductiveType`, `Constructor`) have only the initials capitalized. The remaining tags (`arity`, `source`, `target`) are only "syntactic sugar". The meaning of each tag is clearly understandable to people acquainted with CIC.

The last level is the *level of theories* which is completely independent from the particular logical framework. In our idea, a theory is a (structured) mathematical document containing objects taken almost freely from different sections. Writing a new theory should consist in developing new objects and assembling these new objects and older ones into the mathematical document. It is during the creation of a theory that objects must also be assigned the particular, semantical, meaning used to classify them, for example into lemmas, conjectures, corollaries, etc. Obviously, each theory, that is exported to a different XML file, does not include the objects directly, but refers to them via their URIs.

Theory files have also sections delimiting the scope of variable declarations: to include (a reference to) a definition D depending on a variable V when both reside in a section (directory) R, you must open, inside the theory file, a section referring to R and put inside it the references to V and R. A very small example of a theory file will clarify the above statement:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Theory SYSTEM "http://localhost:8081/getdtd?url=maththeory.dtd">
<Theory uri="cic:/coq/INIT/Logic">
 <!-- Require Export Datatypes -->
 <DEFINITION uri="True.ind"/>
 <DEFINITION uri="False.ind"/>
 <DEFINITION uri="not.con"/>
 <SECTION uri="Conjunction">
  <DEFINITION uri="and.ind"/>
  <VARIABLE uri="A.var"/>
  <VARIABLE uri="B.var"/>
  <THEOREM id="id1" uri="proj1.con"/>
  <THEOREM id="id2" uri="proj2.con"/>
 </SECTION>
 <SECTION uri="Disjunction">
  <DEFINITION uri="or.ind"/>
 </SECTION>
</Theory>
```

All the URIs, but that of `Theory`, are relative URIs; so, the absolute URI of `id1` is "`cic:/coq/INIT/Logic/Conjunction/id1`". In the example you can also see the usage of sections to bound the scope of variables: the scope of `A` and `B` is the section `Conjunction`.

It is important to note that at the theory level, sections are not used to structure the document into, for instance, chapters, paragraphs and so on; many kind of (XML) markup languages have just been developed to do so. Accordingly to the spirit of XML, our theory markup will be freely and modularly intermixed with other kinds of markup, such as XHTML[3]; so, our language will play for mathematical theories the same role of MathML for mathematical expressions and SVG[4] for vectorial graphics. The added value of using the theory level (instead of directly embedding the object markup) is that, while enriching the semantics of the objects of the previous level, it could also be used to enforce some constraints as, for example, on the scope of variables or on the links between theorems and lemmas.

[3] http://www.w3.org/TR/xhtml1
[4] http://www.w3.org/TR/SVG

The module is full working and has been used to export the whole library provided with the Coq System, yielding about 64 Mb of XML (2 Mb after compression). All the obtained XML documents are valid with respect to the DTDs developed for the three levels; we cannot show the DTDs here because of lack of space.

5 Type-Checker

In order to verify that all the needed information was exported from Coq, we have developed a stand-alone type-checker for CIC objects, similar to the Coq one, but fairly simpler and smaller thanks to its independence from the proof engine. The type-checker is now almost finished, lacking only the management of universes. It is the first example of a tool working directly on the XML encoding.

With respect to other type-checkers (as the one of Coq), it is fairly standard but for the peculiar management of the environment: usually, the type-checkers are used to check whole theories, i.e. sequence of definitions or proofs. Each time a definition is checked, it is added to the environment and then it is used in subsequent type-checkings. So, every theorem is always checked with the same, statically defined environment[5]. Our type-checker, instead, is also used to check single objects in an environment that could not yet have the definitions required (e.g. the empty environment). In this case, the environment (a cache, actually) is built "on-demand" during the type-checking of the object: every time a reference to another object not present in the environment is found, the type-checking is interrupted, processing the new object first. Checks are introduced in order to avoid cycles in the definitions, corresponding to an inconsistent environment. In order to make the user understand the strange behaviors described in note 5, that could in theory appear more often in our model, we advocate a way for the user to ask the system what is the inferred universe level of each type.

6 XSL Transformations and MathML

XSLT [7] is a language for transforming XML documents into other XML documents. In particular, a stylesheet is a set of rules expressed in XSLT to transform the tree representing a XML document (the Document Object Model, DOM [1]) into a result tree. When a pattern is matched against elements in the source tree, the corresponding template is instantiated to create part of the result tree. In this way the source tree can be filtered and reordered, and arbitrary structure can be added. A pattern is an expression of XPath [6] language, that allows to match elements according to their values, structure and position in the source tree.

XSLT is primarily aimed to associate a *style* to a XML document, generating formatted documents suitable for rendering purposes. Once XML is chosen as the data description language to encode the mathematical information, it is quite natural to use stylesheets as the standard mechanism to automatically generate the associated notation from a XML mathematical document.

In the same way MathML can naturally be chosen as the target formatting language for mathematics. MathML [3] is an instance of XML for describing the notation of a mathematical expression, capturing both its structure and content. MathML has, roughly, two categories of markup elements: the presentation markup, which can be used to describe the layout structure of mathematical notation, and the content markup, whose aim is to provide an explicit encoding of the *underlying* mathematical structure of an expression.

[5] This is almost true: a user is free to load two theories in any order, changing in this way the environment used during the second type-checking. Sometimes this can lead to strange behaviors, i.e. two theories that are correct if loaded alone could not be if loaded together. This phenomenon is due to the type-inference of the universe level and in particular to the creation of cycles between the universe level constraints.

Although the target of MathML is the encoding of expressions (so it cannot describe mathematical objects and documents), the use of MathML presentation markup as a privileged rendering format is clearly justified by the fact of providing a standard way to enable mathematical expressions to be served, received and processed on the world wide web. Moreover, its presentation part has been already implemented by several applications.

Also, the choice of using MathML content markup as an intermediate representation between the logic-dependent representation of the mathematical information and its presentation is justified by several reasons:

- Even if the content markup is restricted to the encoding of a particular set of formulas (the ones used until the first two years of college in the United States), it is essentially extensible and flexible[6].
- Passing through this semi-formal representation will improve the modularity of the overall architecture: many specific logical dialects can be mapped into the same intermediate language (or into suitable extensions of it). Moreover several stylesheets can be written for this single intermediate language to generate specific rendering formats.
- The characteristic of portability of MathML content markup can be exploited for instance when cutting and pasting terms from an application to another.
- This content level simplifies the structure of a CIC term: there is no more syntactic sugar, but only "pure expressions".
- We can capture the semantics of well-known terms, as for example the disjunction, marking them with the corresponding content elements (e.g. `or`).

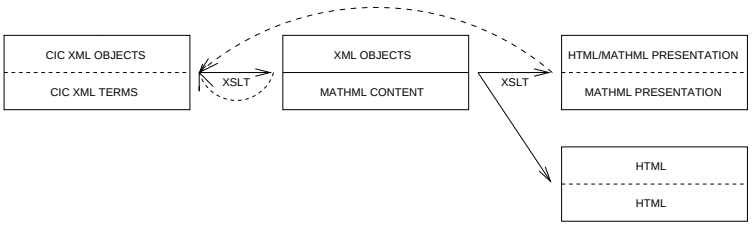

Figure 2: Transformations on the first two levels of CIC XML files: the backward arrows represent links from the content and presentation files to the corresponding CIC XML files.

As you can see in Figure 2, there are two phases of stylesheets application: the first generates the content representation from the CIC XML one; the second generates from this intermediate representation two (and eventually several other) possible kinds of output format, either the MathML presentation markup or the HTML markup. We can immediately note that MathML content can only describe CIC terms (expressions) and so we have had to add a second language to describe CIC objects in the intermediate step. Every obtained document is valid with respect to the corresponding DTD developed for it; in particular, we use the DTD recommended by the last MathML specification.

This is an example of content markup[7]:

[6] The most important element for extensibility purposes is `csymbol`, defined for constructing a symbol whose semantics is not part of the core content elements provided by MathML, but defined externally.

[7] This fragment belongs the example of content file in Appendix A.

```
<m:apply>
 <m:csymbol>app</m:csymbol>
 <m:ci definitionURL="cic:/coq/INIT/Logic/Conjunction/and_ind.con">and_ind</m:ci>
 <m:ci>A</m:ci>
 <m:ci>B</m:ci>
 <m:ci>A</m:ci>
 <m:lambda>
  <m:bvar><m:ci>H0</m:ci><m:type><m:ci>A</m:ci></m:type></m:bvar>
  <m:lambda>
   <m:bvar><m:ci>H1</m:ci><m:type><m:ci>B</m:ci></m:type></m:bvar>
   <m:ci>H0</m:ci>
  </m:lambda>
 </m:lambda>
 <m:ci>H</m:ci>
</m:apply>
```

During the realization of the content stylesheets, we have had to face and solve several problems connected to the MathML specification:

- Objects in general, cannot be considered as terms. Hence, we have added to the MathML content markup a XML level to describe the level of objects (the mixing will be possible using the W3C namespaces [4]). Anyway, this language is still dependent on the particular logical framework.
- We need to semantically describe in content markup not only entities defined in CIC, as the conjunction, but also the primitive CIC operators, as the application or the lambda abstraction. When encoding the entities, MathML markup could be really exploited. To preserve their formal semantics, we keep pointers to the XML files of their CIC definitions. With regard to most CIC operators, there are no specific MathML content markup elements. To solve this problem we use the `csymbol` element. In particular, we have chosen to introduce also a `csymbol` for the CIC application (see the example above) instead of using the content element `apply`. The MathML application is the general way of building up a mathematical expression, and so it is different from the CIC application. We want to make a clean semantical distinction between the logical application between two terms, and the "application" of some operators (like the sin function) to its arguments.

As you can see in Figure 2, we produce MathML content and presentation in two distinct steps. The only way to combine and link together content and presentation in compliance to the MathML specification consists of using the `semantics` element. This content element is quite ambiguous, a kind of "bridge" between content and presentation; moreover, it is currently ignored by all the browsers supporting MathML. For us, a natural improvement should consist of having content and the associated presentations in different files, one for the content expression and one for each presentation. Then we need to relate a presentation expression and subexpressions to the respective content expression and subexpressions. This can be achieved in a standard way using the machinery of XLink [5].

The above solution can also be exploited for the implementation of the links of Figure 2 for linking the content and presentation markup to the corresponding source CIC XML terms. In this way the user can browse the MathML presentation and also modify it: the changes will have effect on the corresponding CIC XML file.

An example of MathML presentation markup generated after the second phase is[8]:

[8] This fragment belongs the the example of content file in Appendix A.

```
<m:mrow>
 <m:mo stretchy="false">(</m:mo>
 <m:mi>and_ind</m:mi>
 <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mi>A</m:mi>
 <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mi>B</m:mi>
 <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mi>A</m:mi><m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mrow>
  <m:mo color="Red">&lambda;</m:mo>
  <m:mi>H0</m:mi>
  <m:mo>:</m:mo>
  <m:mi>A</m:mi>
  <m:mo>.</m:mo>
  <m:mrow>
   <m:mo color="Red">&lambda;</m:mo>
   <m:mi>H1</m:mi>
   <m:mo>:</m:mo>
   <m:mi>B</m:mi>
   <m:mo>.</m:mo>
   <m:mi>H0</m:mi>
  </m:mrow>
 </m:mrow>
 <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mi>H</m:mi>
 <m:mo stretchy="false">)</m:mo>
</m:mrow>
```

To generate the presentation markup from the corresponding content markup, we use, among others, a stylesheet, compliant with the last specification of MathML, written by Igor Rodionov[9]. This stylesheet, written in collaboration with the MathML Working Group, transforms MathML presentation markup in MathML content one. Here, we want to stress that the possibility to re-use work done by other people is an essential aspect of the our general methodology of work.

We have had to solve several problems regarding the presentation output:

- We have had to associate an output to every object and to every `csymbol` defined in the content phase.
- We have modified the MathML stylesheet to implement the policy of line-breaking and alignment for long terms: our choice consists of using tables made of multiple rows. The `mtable` element is specifically designed to arrange expressions in a bi-dimensional layout and in particular it provides a set of related elements and attributes to achieve proper alignment and line-breaking. Our policy consists of breaking expressions only in specific points and only when the row length exceeds a threshold (that could make the reading difficult), generating tables for alignment purposes.

MathML is not the only format exploited: another presentation format is HTML, due to the wide-spreading of browsers for it and its native hypertextual nature. Thanks to the modular architecture (see Figure 2), many others could be added too.

[9]Computer Science Department of the University of Western Ontario, London, Canada.

We will exploit the same modular architecture of the object level at the level of theories. At this level we can use the same presentation formats of the previous levels; on the contrary, there is no standard language for describing theories at the content level. So we will develop a new (XML) language that will be largely independent from the specific foundational dialect and could aspire to become a standard in the same way MathML is for mathematical expressions.

7 Interfaces to HELM

Two of the main goals of the project are the easiness in augmenting and browsing the library:

1. Every user with a small amount of http or FTP space should be able to publish a document.
2. Every user with a common browser should be able to browse the library.

To fulfill these aims, we must face the actual state of technology:

1. Currently, almost all of the Internet users have a web space, but usually without being allowed to run any kind of program on the server, even simple CGIs. So no intelligence can be put on the publisher side.
2. The browser technology is rapidly evolving in such a way that we can expect in a few time to have browsers able to understand MathML and, probably, even to apply XSLT stylesheets. At the same time, though, if we require the browser to be standard, then we have to put the intelligence on the other side, i.e. on the server.

Therefore, where can we put the intelligence? A first answer is the creation of *presentation sites* able to get documents from distribution sites, process them (e.g. applying stylesheets) and return them to the users in the user requested format. We have been able to create presentation sites based on Cocoon[10], a XML server-based web-publishing framework. In a future work ([8]) we will describe in details how we have done this.

Though this solution is perfect for browsing and doing simple elaborations, it gives the user too strict interaction possibilities, which are required for more complex tasks (as the creation of new theories, for example). Hence, more advanced interfaces with such capabilities are required. These interfaces must be run on the user's machine and should, at least, be able to provide all the processing functionalities of the presentation servers, including XSLT stylesheets application. At the same time, they should also overcome the limitations of standard browsers through the addition of new interaction possibilities.

Since our preferential browsing language will be MathML, our interface should at least be able to render its presentation markup. Unfortunately, there are no satisfactory implementations available yet. Moreover, we need also to interact with the MathML rendered files, for example for editing. Not only forms of interaction with this kind of markup have never been proposed before, but we also need to reflect the changes on the source files of the XSLT rendering transformations. This has lead us to the development of a brand new engine with rendering and editing capabilities for documents embedding MathML presentation markup. This engine is designed to be stand-alone and will be made freely available as a Gtk[11] widget.

We have just integrated our widget, the type-checker and an external XSLT processor into a minimal interface that we are going to extend with editing functionalities.

[10] http://xml.apache.org/cocoon
[11] http://www.gtk.org

8 The model of distribution

Mathematical documents have some peculiar properties. First of all a mathematical document should be immutable: the correctness of a document A that refers to a document B can be guaranteed only if B does not change. Notwithstanding this, new versions of a mathematical document could be released (for example if a conjecture is actually proved). Secondly, a user cannot be forced to retain a copy of his document forever, even if other documents refer to it. So, it should be possible for everyone to make a copy of a document and also distribute it. When more than a copy is available, the user could be able to download it from a particular server (for example, from the nearest one). This implies that documents could not be referred to via URLs, but only with logical names in the form of URIs. A particular naming policy should then be adopted to prevent users to publish different documents under the same URI.

To fulfill these requirements, we have adopted almost the same model of distribution of the Debian packaging system APT[12] that has similar requirements (a package could not be modified, but only updated, it is available on different servers, could be downloaded from the user preferred server).

Every document is identified by an URI. "`cic:/coq/INIT/Datatypes/nat.ind`" is an example of such an URI that references an inductive definition (".ind") in the subsection `Datatypes` of the subsection `INIT` of the section `coq`.
Similarly, the URI "`theories:/coq/INIT/Datatypes.theory`" refers to the mathematical theory named `Datatypes` located in the subdirectory `INIT` of the directory `coq`.

In order to translate the URI to a valid URL, a particular tool, named *getter*, is needed. It takes in input an ordered list of servers and an URI and returns the URL of the document on the first server in the list providing it. In order to know which documents a server provides, each server publishes a list of the URIs of its documents with the respective URLs.

During the processing of an XSLT stylesheet, the processor must be able to open some documents, i.e. it should be able to ask the getter to resolve the URIs it needs. There is no standard way to tell the XSLT processor how to resolve URIs or concatenate external programs to do this. So, we have implemented the getter as an HTTP proxy-daemon reachable through a known URL that takes the URI as a CGI parameter, downloads the document using the resolved URL and returns it. An example of the syntax we are currently using to contact the getter is "`http://phd.cs.unibo.it:8081/get?uri="cic:/coq/INIT/Datatypes/nat.ind"`"

If the getter resides on the user machine, then the downloaded document could be cached for improving performances. Once cached, it could also be added to the list of documents the server has. In such a way, often referred to or simply interesting documents fast widespread over the net, downloading times are reduced and the author can freely get rid of his copy of the document if he needs it no more.

This architecture imposes no constraints on the naming policy: up to now we have not chosen or implemented one yet. To face the issue, one possibility can be the choice of having a centralized naming authority, even if other more distributed scenarios will surely be considered.

9 Further Developments

We are soon going to develop:

[12] http://www.debian.org

Tools for indexing and retrieval of mathematical documents, based on meta-data specified in the Resource Description Framework (RDF)[13]. RDF uses XML to define a foundation for processing meta-data, complements XML and provides interoperability between applications that exchange machine-understandable information on the web.

Tools for the (re)annotation of mathematical objects and terms: the intuitive meaning of these entities is usually lost in their description in a logical framework. Even their automatically extracted presentations in a natural language are often unsatisfactory, being quite different from the typical presentation in a book. We believe that a feasible solution is giving the user the possibility of enriching terms with annotations given in an informal, still structured language.

10 Conclusions

In this paper we have presented the current status of the HELM project, whose aim is to exploit the potentiality of XML technology for the creation and maintenance of an electronic, distributed, hypertextual library of formal mathematical knowledge.

Our ultimate goal is the extension of the library to other logical frameworks and systems. This will also be an important test bench for the whole architecture.

Another fundamental improvement would be the development of new modular proof engines, supporting step-by-step informal annotations on proofs in natural language (see [8] for a deeper discussion of this topic).

References

[1] Document Object Model (DOM) Level 2 Specification. Version 1.0, W3C Candidate Recommendation, 10 May 2000.
`http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510/`

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

[3] Mathematical Markup Language (MathML) 2.0 W3C Working Draft, 28 March 2000.
`http://www.w3.org/TR/2000/WD-MathML2-20000328/`.

[4] Namespaces in XML, W3C Recommendation, 14 January 1999.
`http://www.w3.org/TR/REC-xml-names/`

[5] XML Linking Language (XLink), W3C Working Draft (last call), 21 February 2000.
`http://www.w3.org/TR/xlink/`

[6] XML Path Language (XPath) Version 1.0, W3C Recommendation, 16 November 1999.
`http://www.w3.org/TR/xpath`

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

[8] Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., "XML, Stylesheets and the re-mathematization of Formal Content", Department of Computer Science, Bologna, Italy, May 2000.

[13] http://www.w3.org/TR/REC-rdf-syntax.

[9] B. Barras et al., "The Coq Proof Assistant Reference Manual, version 6.3.1", http://pauillac.inria.fr/coq/

[10] Berners-Lee, T., "Universal Resource Identifiers in WWW", RFC 1630, CERN, June 1994.

11 APPENDIX A

Cic Xml file:

```
<!DOCTYPE HTML SYSTEM "http://localhost:8081/getdtd?url=cic.dtd">
<Definition name="proj1" params="0: A B">
  <body>
   <LAMBDA>
     <source>
       <APPLY>
         <MUTIND notype="0" uri="cic:/coq/INIT/Logic/Conjunction/and.ind"/>
         <VAR reluri="0,A"/>
         <VAR reluri="0,B"/>
       </APPLY>
     </source>
     <target binder="H">
       <APPLY>
         <CONST uri="cic:/coq/INIT/Logic/Conjunction/and_ind.con"/>
         <VAR reluri="0,A"/>
         <VAR reluri="0,B"/>
         <VAR reluri="0,A"/>
         <LAMBDA>
           <source><VAR reluri="0,A"/></source>
           <target binder="H0">
             <LAMBDA>
               <source><VAR reluri="0,B"/></source>
               <target binder="H1"><REL binder="H0" value="2"/></target>
             </LAMBDA>
           </target>
         </LAMBDA>
         <REL binder="H" value="1"/>
       </APPLY>
     </target>
   </LAMBDA>
  </body>
  <type>
   <PROD>
     <source>
       <APPLY>
         <MUTIND notype="0" uri="cic:/coq/INIT/Logic/Conjunction/and.ind"/>
         <VAR reluri="0,A"/>
         <VAR reluri="0,B"/>
       </APPLY>
     </source>
     <target><VAR reluri="0,A"/></target>
   </PROD>
  </type>
</Definition>
```

Corresponding content file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Definition name="proj1" xmlns:m="http://www.w3.org/1998/Math/MathML">
  <Params>0: A B</Params>
  <body>
   <m:math>
    <m:lambda>
     <m:bvar>
      <m:ci>H</m:ci>
      <m:type>
       <m:apply>
        <m:and definitionURL="cic:/coq/INIT/Logic/Conjunction/and.ind"/>
        <m:ci>A</m:ci>
        <m:ci>B</m:ci>
       </m:apply>
      </m:type>
     </m:bvar>
     <m:apply>
      <m:csymbol>app</m:csymbol>
      <m:ci definitionURL="cic:/coq/INIT/Logic/Conjunction/and_ind.con">and_ind</m:ci>
      <m:ci>A</m:ci>
      <m:ci>B</m:ci>
      <m:ci>A</m:ci>
      <m:lambda>
       <m:bvar><m:ci>H0</m:ci><m:type><m:ci>A</m:ci></m:type></m:bvar>
       <m:lambda>
        <m:bvar><m:ci>H1</m:ci><m:type><m:ci>B</m:ci></m:type></m:bvar>
        <m:ci>H0</m:ci>
       </m:lambda>
      </m:lambda>
      <m:ci>H</m:ci>
     </m:apply>
    </m:lambda>
   </m:math>
  </body>
  <type>
   <m:math>
    <m:apply>
     <m:csymbol>arrow</m:csymbol>
     <m:apply>
      <m:and definitionURL="cic:/coq/INIT/Logic/Conjunction/and.ind"/>
      <m:ci>A</m:ci>
      <m:ci>B</m:ci>
     </m:apply>
     <m:ci>A</m:ci>
    </m:apply>
   </m:math>
  </type>
</Definition>
```

Corresponding presentation file:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">
<m:math xmlns:m="http://www.w3.org/1998/Math/MathML">
  <m:mtable align="baseline 1" columnalign="left" equalrows="false">
   <m:mtr>
    <m:mtd><m:mrow><m:mtext>DEFINITION proj1() OF TYPE</m:mtext></m:mrow></m:mtd>
   </m:mtr>
   <m:mtr>
    <m:mtd>
     <m:mrow>
      <m:mphantom><m:mtext>__</m:mtext></m:mphantom>
      <m:semantics>
       <m:mrow>
        <m:mo stretchy="false">(</m:mo>
        <m:mrow>
         <m:mi>A</m:mi>
         <m:mo><mchar name="wedge"></mchar></m:mo>
         <m:mi>B</m:mi>
        </m:mrow>
        <m:mo color="Blue">&rarr;</m:mo>
        <m:mi>A</m:mi>
        <m:mo stretchy="false">)</m:mo>
       </m:mrow>
       <m:annotation-xml encoding="MathML">
        <m:apply>
         <m:csymbol>arrow</m:csymbol>
         <m:apply>
          <m:and definitionurl="cic:/coq/INIT/Logic/Conjunction/and.ind"/>
          <m:ci>A</m:ci>
          <m:ci>B</m:ci>
         </m:apply>
         <m:ci>A</m:ci>
        </m:apply>
       </m:annotation-xml>
      </m:semantics>
     </m:mrow>
    </m:mtd>
   </m:mtr>
   <m:mtr><m:mtd><m:mrow><m:mtext>AS</m:mtext></m:mrow></m:mtd></m:mtr>
   <m:mtr>
    <m:mtd>
     <m:mrow>
      <m:mphantom><m:mtext>__</m:mtext></m:mphantom>
      <m:semantics>
       <m:mrow>
        <m:mo color="Red">&lambda;</m:mo>
        <m:mi>H</m:mi>
        <m:mo>:</m:mo>
        <m:mrow>
         <m:mi>A</m:mi>
         <m:mo><mchar name="wedge"></mchar></m:mo>
         <m:mi>B</m:mi>
        </m:mrow>
        <m:mo>.</m:mo>
        <m:mrow>
         <m:mo stretchy="false">(</m:mo>
         <m:mi>and_ind</m:mi>
         <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mi>A</m:mi>
         <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mi>B</m:mi>
         <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mi>A</m:mi><m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mrow>
          <m:mo color="Red">&lambda;</m:mo>
          <m:mi>H0</m:mi>
          <m:mo>:</m:mo>
          <m:mi>A</m:mi>
          <m:mo>.</m:mo>
           <m:mrow>
            <m:mo color="Red">&lambda;</m:mo>
            <m:mi>H1</m:mi>
            <m:mo>:</m:mo>
            <m:mi>B</m:mi>
            <m:mo>.</m:mo>
            <m:mi>H0</m:mi>
```

```
          </m:mrow>
         </m:mrow>
         <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mi>H</m:mi>
         <m:mo stretchy="false">)</m:mo>
        </m:mrow>
       </m:mrow>
       <m:annotation-xml encoding="MathML">
        <m:lambda>
         <m:bvar>
          <m:ci>H</m:ci>
          <m:type>
           <m:apply>
            <m:and definitionurl="cic:/coq/INIT/Logic/Conjunction/and.ind"/>
            <m:ci>A</m:ci><m:ci>B</m:ci>
           </m:apply>
          </m:type>
         </m:bvar>
         <m:apply>
          <m:csymbol>app</m:csymbol>
          <m:ci definitionurl="cic:/coq/INIT/Logic/Conjunction/and_ind.con">and_ind</m:ci>
          <m:ci>A</m:ci>
          <m:ci>B</m:ci>
          <m:ci>A</m:ci>
          <m:lambda>
           <m:bvar><m:ci>H0</m:ci><m:type><m:ci>A</m:ci></m:type></m:bvar>
           <m:lambda>
            <m:bvar><m:ci>H1</m:ci><m:type><m:ci>B</m:ci></m:type></m:bvar>
            <m:ci>H0</m:ci>
           </m:lambda>
          </m:lambda>
          <m:ci>H</m:ci>
         </m:apply>
        </m:lambda>
       </m:annotation-xml>
      </m:semantics>
     </m:mrow>
    </m:mtd>
   </m:mtr>
  </m:mtable>
</m:math>
```