HELM and the Semantic Math-Web

Andrea Asperti, Luca Padovani, Claudio Sacerdoti Coen, and Irene Schena

Department of Computer Science
Via di Mura Anteo Zamboni 7, 40127 Bologna, Italy
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. By the application of XML technology to the large repositories of structured, content oriented information offered by Logical Frameworks we meet the ultimate goal of the Semantic Web, that is to allow machines the sharing and exploitation of knowledge in the Web way, i.e. without central authority, with few basic rules, in a scalable, adaptable, extensible manner.

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, the recent emphasis on *content* and the new frontier of the so called "Semantic Web.[1]" The goal of the Semantic Web is to pass from a "machine readable" to a "machine understandable" encoding of the information: establishing a layer of machine understandable data would allow automated agents, sophisticated search engines and interoperable services and will enable higher degree of automation and more intelligent applications.

In contrast with current encodings of mathematical information (e.g. in digital libraries), which are "machine-readable" but not "machine-understandable", Logical Frameworks offer *huge* repositories of structured, content oriented information, naturally providing a major arena for the Semantic Web and its technologies. The point is to allow access to this mathematical knowledge in the

[1] Semantic Web Activity, http://www.w3.org/2001/sw.

R.J. Boulton and P.B. Jackson (Eds.): TPHOLs 2001, LNCS 2152, pp. 59–74, 2001.

"Web way", i.e. without central authority, with few basic rules, in a scalable, adaptable, extensible manner.

The first, mandatory step in this direction is the direct encoding of the libraries in XML, which has recently imposed as the main tool for representation, manipulation, linking and exchange of structured information. But of course the broad goal of the Semantic Web goes far beyond the trivial suggestion to adopt XML as a neutral specification language for the "compiled" versions 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. Here is a short list of the added-value offered by the XML approach:

Standardization. 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 of the specific logical system.

Publishing. XML offers sophisticated web-publishing technologies (Stylesheets, MathML, ...) which can be profitably used to solve, in a *standard* way, the annoying notational problems that traditionally afflict formal mathematics.

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

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.

Modularity. The "XML-ization" process should naturally lead 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. This is the new *content-based* architecture of future systems.

In this article we present our project on the use of XML technology for the development and maintenance of distributed repositories of formal mathematical knowledge: the Hypertextual Electronic Library of Mathematics (HELM[2]).

2 The eXtensible Markup Language

Perhaps, the best way to introduce XML in few lines is to take a look at a simple example. The following XML document is a possible encoding of the definition of the inductive type of natural numbers in Coq [3].

[2] `http://www.cs.unibo.it/helm`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE InductiveDefinition
  SYSTEM "http://www.cs.unibo.it/helm/dtd/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>
```

XML gives a method for structuring 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 `InductiveDefinition` or `InductiveType` (they can contain other elements or text), or they are empty elements, as `SORT` or `REL`. The previous example contains no text: this is a peculiarity of really *formal* encodings, where every bit of knowledge has a specific intelligence worth to be encapsulated in markup.

The XML specification defines a XML document to be well-formed if it meets some syntactical constraints on 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 labeled brackets for structuring the document.

XML 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, here is a DTD fragment for the previous XML file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
...
<!ENTITY % term '(LAMBDA|CAST|PROD|REL|SORT|APPLY|VAR|META|IMPLICIT|
                  CONST|LETIN|MUTIND|MUTCONSTRUCT|MUTCASE|FIX|COFIX)'>
...
<!ELEMENT InductiveDefinition (InductiveType+)>
<!ATTLIST InductiveDefinition
          noParams NMTOKEN #REQUIRED
          params   CDATA   #REQUIRED
          id       ID      #REQUIRED>
<!ELEMENT InductiveType (arity,Constructor*)>
<!ATTLIST InductiveType
          name      CDATA        #REQUIRED
```

```
          inductive (true|false) #REQUIRED>
<!ELEMENT Constructor %term;>
<!ATTLIST Constructor name CDATA #REQUIRED>
...
```

A `InductiveDefinition` element may contain one or more `InductiveType` elements, each `InductiveType` contains an `arity` followed by a possibly empty sequence of `Constructor`s, and a `Constructor` is an arbitrary `term` (an `ENTITY` is a macro declaration). The attributes of `InductiveDefinition` are `noParams`, `params` and `id`, while `name` is the only attribute of `Constructor`. The keyword `REQUIRED` states that an attribute 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 [6]. A generic URI is made of a formatted (structured) string of characters whose intended meaning is associated by the applications managing it. URLs (Uniform Resource Locators) are a particular kind of URIs specifically designed to name resources accessed by means of a given standard protocol (for example the HTTP protocol). URLs consist of a first part identifying the protocol and a host followed by a second part to locate the resource on it.

URLs can be resolved by standard processing tools and browsers, but suffer from problems of consistency: moving the target document leads to dangling pointers; moreover, being physical names, they cannot be used to identify a whole set of copies located on different servers for fault-tolerance and load-balancing purposes. URIs, instead, can be designed as logical names, leaving to applications the burden of resolution to physical names. So, for examples, the URI "`cic:/Coq/Reals/Rdefinitions/R.con`" could be used as a logical name for the axiom which defines the existence of the set `R` of real numbers in the standard library of the Coq Proof Assistant; then, an application is required to map the URI to a physical name (an URL) as

`http://coq.inria.fr/library/Reals/Rdefinitions/R.con.xml`

3 The HELM Project

The overall architecture of our project is depicted in Fig. 1.

Once XML has been chosen as the standard encoding format[3] 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 [3]. In the near future, we expect that similar exporting functionalities will be provided by the developers of the other logical systems. We will describe the exporting issues in Sect. 4.

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

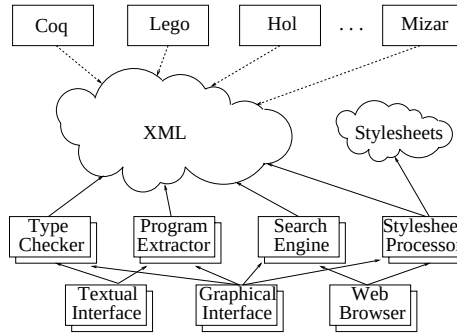

Fig. 1. Architecture of the HELM Project.

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 Sect. 5 we shall briefly discuss our implementation of a type-checking tool, while in Sect. 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 Sect. 7.

Because of the particular nature of the library, we have also provided a suitable model of distribution, which is discussed in more detail in Sect. 8.

4 Exporting from Coq

Coq is one of the most advanced proof-assistants, 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, whose last stable release is V6.3.1, very big and complex. In fact, the only practical way to work with the information encoded in Coq is that of writing a new module that extends the system gaining access to its internal representation.

A new release of Coq, called V7, is being now developed with the precise aim to reduce the implementation complexity; notwithstanding this, finding the right information inside the system itself is not a trivial task: first, information is encoded in Coq's data structures which change among different versions of the system; second, the required information is often not directly available. For

example, when a file is loaded in V6.3.1, its path is forgotten, while we would like to export files in directories reflecting the logical structure of the library.

In the rest of this section we only focus on the implementation of the exporting module[4] for version 6.3.1 which is a representative example of the difficulties that could be faced when exporting from an existing proof-assistant without the collaboration of its development team.

4.1 Relevant Information to Be Exported

To design the module, the first difficulty has been the identification of which information is worth to be exported. 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 cannot 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 λ-term of the proof; hence it is possible to add as an attribute to a sub term 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, surely useless for browsing purposes or for indexing, but necessary to every proof-checker implementation. Other times, instead, the added information is important only for browsing purposes. For example, sometimes the user asks the system to infer a type and probably does not want to view the inferred type thereafter.

So, we have decided to export a first core of information, roughly corresponding to the only one available at the logical level, according to a *principle of minimalism*: *no redundant information should be exported*. If the principle were not followed, in every application loading an XML file we would have to add checks to verify the consistency of the redundant (and possibly useless!) information. In Sect. 4.2 we will see how this does not prevent us to link in a precise way additional information to the core one.

[4] The module is fully 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).

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) can never appear alone, but only as part of an object definition. In Coq, the terms are CIC λ-expressions, i.e. variables (encoded as DeBrujin indexes), λ-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 co-inductive function definitions. The whole level is extremely dependent on 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 stored 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 sub-theories. In HELM, the former usage is retained, while theories are described in another way (see the third level). In Coq, the objects are constants, representing definitions, theorems and axioms, (the former two have both a type and a body, while the latter has a type only), variables, (co)inductive definitions (such as `nat` for the natural numbers) and proofs in progress.

The last level is the *level of theories* which is completely independent of 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. Theories, that are described in different XML files, do not include the objects directly, but refers to them via their URIs.

The following is an example of theory file with two sections, the first one delimiting the scope of variables A and B.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Theory SYSTEM "http://www.cs.unibo.it/helm/dtd/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/proj1.con`". 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;[5] so, our language will play for mathematical theories the same role of MathML for mathematical expressions and SVG[6] 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.

4.2 Auxiliary Information and Metadata

In this paragraph we address the problem of the association of additional and possibly redundant information to the one exported using the Coq module. The purpose of this added information is to enable or facilitate specific functionalities such as rendering, searching, indexing and so on.[7]

A simple observation suggests that such information could be naturally associated either to the whole object definition (e.g. the name of the author of a theorem) or to a particular node of a λ-term (e.g. the tactic used to generate it or an informal description of its meaning). So, we can easily store the additional information in a distinct XML file and use XLink technology to relate it to the appropriate XML element in the corresponding logical-level document. Moreover, in the specific case of metadata, we can benefit from the Resource Description Framework (RDF, [11] [12]), which provides a general model for representing metadata as well as a syntax for encoding and exchanging this metadata over the Web. In several cases, this meta-information can be extracted from the document, for instance by means of XSL Transformations. It is important to note that, in this approach, no modification at all is required to the DTDs or to the source document. As a side effect, an application can consult just the XML files holding the information it really needs, without having to parse, check for consistency and finally ignore non-interesting information.

[5] `http://www.w3.org/TR/xhtml1`.

[6] `http://www.w3.org/TR/SVG`.

[7] For instance, a major example of such additional information, which is essential for rendering purposes, are the intermediate conclusions inside complex proofs (see [4]), which are typically omitted (as redundant) in a Curry-Howard representation of proofs as λ-terms.

5 Proof-Checker

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

With respect to other proof-checkers (as the one of Coq), it is fairly standard but for the peculiar management of the environment: usually, proof-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-checking. So, every theorem is always checked with the same, statically defined environment. Our proof-checker, instead, builds the environment (a cache, actually) "on-demand": 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.

6 XSL Transformations and MathML

XSLT [17] 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 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 [16] 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, thus providing a standard tool for processing and transforming XML mathematical document, according to alternative notations and encodings.

A natural encoding for mathematics on the Web is MathML [8], 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 describes the layout structure of mathematical notation, and the content markup, that provides an explicit encoding of the *underlying* mathematical structure of an expression.

Although the target of MathML is the encoding of expressions (so it cannot describe mathematical objects and documents), the MathML Presentation Markup can be considered as a privileged rendering format, providing a standard on the web already implemented by several applications.

The choice of 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 extensible and flexible.[8]
- 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 cutting and pasting terms from an application to another.
- MathML content markup can capture the usual *informal* semantics of well-known operators, as for example the disjunction, marking them with the corresponding content elements (e.g. `or`). Their different formal content is preserved by means of backward pointers to the low level specification.

A feasible alternative to MathML content is provided by OpenMath[9] (see the MathML [8] recommendation for a discussion about the two approaches).

As depicted in Fig. 2, there are two sequential phases of stylesheets application: the first one generates the content representation from the CIC XML one; the second one generates either the MathML presentation markup or the HTML markup (and possibly others presentational formats as well).

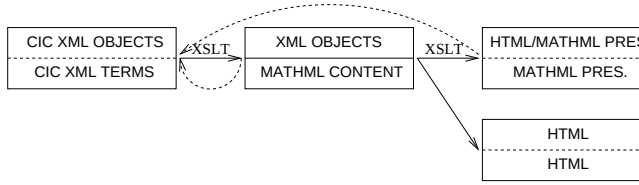

Fig. 2. Transformations of CIC XML Files: The backward arrows represent links from the content and presentation files to the corresponding CIC XML files.

The following is an example of content markup

```
<apply>
 <csymbol>app</csymbol>
 <ci definitionURL=
      "cic:/Coq/Init/Logic/Conjunction/and_ind.con">and_ind</ci>
 <ci>A</ci>
 <ci>B</ci>
```

[8] 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.

[9] `http://www.openmath.org`.

```
 <ci>A</ci>
 <lambda>
  <bvar><ci>H0</ci><type><ci>A</ci></type></bvar>
  <lambda>
   <bvar><ci>H1</ci><type><ci>B</ci></type></bvar>
   <ci>H0</ci>
  </lambda>
 </lambda>
 <ci>H</ci>
</apply>
```

The main problems related to the definition of content stylesheets are:

- 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 (using different namespaces [9]).
- Operators which are not covered by the MathML content base set are encoded by means of the `csymbol` element.
- The standard semantics of MathML content elements is only informally defined in the specification. In HELM, formal semantics of content elements is enforced, wherever it is possible, by means of pointers to the XML files of their corresponding CIC definitions.

As you can see in Fig. 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, the next natural improvement will 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 and XPointer [14,15].

The above solution has been also exploited for the implementation of the links of Fig. 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:

```
<mrow>
 <mo stretchy="false">(</mo>
 <mi>and_ind</mi><mphantom><mtext>_</mtext></mphantom>
 <mi>A</mi><mphantom><mtext>_</mtext></mphantom>
 <mi>B</mi><mphantom><mtext>_</mtext></mphantom>
 <mi>A</mi><mphantom><mtext>_</mtext></mphantom>
 <mrow>
```

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

To generate presentation markup from the corresponding content markup, we use, among others, a stylesheet, compliant with the last specification of MathML, written by Igor Rodionov.[10] This stylesheet, written in collaboration with the MathML Working Group, transforms MathML content markup in MathML presentation one.

We must 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 linebreaking 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 two-dimensional layout and in particular it provides a set of related elements and attributes to achieve proper alignment and linebreaking.

As we have said above, 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 Fig. 2), many other formats could be added too.

Finally, note that the general architecture gives the possibility of applying stylesheets of arbitrary complexity. For example, we have developed stylesheets to render proofs in natural language, in the spirit of [5].

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 of the specific foundational dialect and could aspire to play the same role MathML plays for mathematical expressions. An already existent corresponding proposal for OpenMath is OMDoc [10]. A possibility could consist of achieving the integration of our proposal with an extension to MathML of OMDoc, giving a common standard way to represent the structure of a mathematical document.

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

7 Interfaces to HELM

Two of the main goals of HELM 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 fullfil 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 expect in a few time to have browsers able to understand MathML and, probably, even to apply XSLT stylesheets. At the moment, 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 retrieve documents from distribution sites, process them (e.g. applying stylesheets) and return them to the users in the user requested format.

In a first prototype of presentation site we relied on the capabilities of Cocoon, a XML server-based web-publishing framework, for the processing of the requested documents. Recently we adopted a more modular and more reusable solution, consisting in having a completely separate component, a servlet indeed, for the application of stylesheets. This component is designed to receive HTTP requests of stylesheet application, sending the result of the transformation as the reply of the request. In this way, ordinary browsing tools can access the processed documents simply by invoking the servlet with a proper URL containing both a reference to the source document to be processed, the list of stylesheets to be applied and possibly a sequence of parameters. In this way the number and order of the stylesheets can be changed at any invocation and their behavior may be affected by the value of the parameters passed by the client. Moreover, the documents, the stylesheets and the DTDs are not constrained to reside on the same host of the processor, thus over-passing the greatest limitation of the Cocoon-based framework.

Though this solution is perfect for browsing[11] 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. At the same time, they should

[11] An on-line library of Coq V7 theories can be found at the address `http://phd.cs.unibo.it/helm/library/`. Each document is rendered on-the-fly either to HTML or to MathML.

also overcome the limitations of standard browsers through the addition of new interaction possibilities.

Since our preferential browsing language is MathML, our interface should at least be able to render its presentation markup. Unfortunately, there are no satisfactory implementations available yet. Moreover, we also need 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 led us to the development of a brand new engine with rendering and editing capabilities for documents embedding MathML presentation markup. The engine, which is also able to export rendered MathML documents to Postscript, is designed to be stand-alone and it is freely available as a Gtk widget.[12]

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

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 should be able to download it from any suitable 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[13] which 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. For example, the URI that references an inductive definition (".ind") in the subsection `Datatypes` of the subsection `Init` of the section `Coq` is "`cic:/Coq/Init/Datatypes/nat.ind`". 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 supplying it. Each server

[12] The widget is now a package of the forthcoming Debian distribution. See `http://www.cs.unibo.it/helm/mml-widget/` for further details.

[13] `http://www.debian.org`.

publishes a list of the URIs of the documents it provides along with their corresponding URL.

In order to flexibly allow communication between the components of HELM architecture, the interface provided by the getter is the one of an HTTP daemon, pre-dating the ideas of the SOAP working draft [13]: when a document is required, its logical name is resolved in an URL, the document is downloaded, it is processed if needed[14] and finally it is returned to the client and it is possibly locally stored for caching purposes. If the machine hosting the getter is also a distribution site, then, once cached, a document could also be added to the list of documents the server exports. In such a way, often referred to or simply interesting documents spread rapidly over the net, downloading times are reduced and the author can freely get rid of his copy of the document if he does not need it any 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 Conclusions and Further Developments

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 large repositories of structured, content oriented information offered by Logical Frameworks, sharing and processing knowledge in compliance with the main philosophy of Web, i.e. without central authority, with few basic rules, in a scalable, adaptable, extensible manner.

Due to our methodology (HELM is independent of Coq), the extension of the library to other logical frameworks and systems does not look problematic.

We are soon going to develop:

Tools for indexing and retrieval of mathematical documents, based on meta-data specified in the Resource Description Framework (RDF, see [11]). RDF provides a foundation for processing meta-data, complements XML, and improves interoperability between applications that exchange 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.

Tools to provide functionalities missing from monolithic proof assistants, such as proof-extraction of proof-editing.

[14] For example, typical processing are deflating compressed files or the resolution of relative URIs contained in the document to absolute ones.

References

1. Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., "Towards a library of formal mathematics". Panel session of the 13th International Conference on Theorem Proving in Higher Order Logics (TPHOLS'2000), Portland, Oregon, USA.
2. 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.
3. Barras, B., et al., "The Coq Proof Assistant Reference Manual, version 6.3.1", `http://pauillac.inria.fr/coq/`.
4. Coscoy, Y., Kahn, G., Thery, L., *Extracting Text from Proofs*. Technical Report RR-2459, INRIA Sophia Antipolis.
5. Coscoy, Y., "Explication textuelle de preuves pour le Calcul des Constructions Inductives", Phd. Thesis, Université de Nice-Sophia Antipolis, 2000.
6. Berners-Lee, T., "Universal Resource Identifiers in WWW", RFC 1630, CERN, June 1994.
7. Extensible Markup Language (XML) (Second Edition). Version 1.0. W3C Recommendation, 6 October 2000. `http://www.w3.org/TR/REC-xml`.
8. Mathematical Markup Language (MathML). Version 2.0. W3C Recommendation, 21 February 2001. `http://www.w3.org/TR/MathML2/`.
9. Namespaces in XML. W3C Recommendation, 14 January 1999. `http://www.w3.org/TR/REC-xml-names/`
10. Open Mathematical Documents (OMDoc) 1.0, November 1 2000. `http://www.mathweb.org/omdoc/`.
11. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`.
12. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation 27 March 2000. `http://www.w3.org/TR/rdf-schema/`.
13. Simple Object Access Protocol (SOAP). Version 1.1. W3C Note, 8 May 2000. `http://www.w3.org/TR/SOAP/`.
14. XML Linking Language (XLink). Version 1.0. W3C Proposed Recommendation, 20 December 2000. `http://www.w3.org/TR/xlink/`.
15. XML Pointer Language (XPointer). Version 1.0. W3C Working Draft (Last Call), 8 January 2001. `http://www.w3.org/TR/xptr/`.
16. XML Path Language (XPath). Version 1.0, W3C Recommendation, 16 November 1999. `http://www.w3.org/TR/xpath`.
17. XSL Transformations (XSLT). Version 1.0, W3C Recommendation, 16 November 1999. `http://www.w3.org/TR/xslt/`.