Mathematical Knowledge Management in HELM

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

Abstract

The paper describes the general philosophy and the main architectural and technological solutions of the HELM Project for the management of large repositories of mathematical knowledge.

The lait-motif is the extensive use of XML-technology, and the exploitation of information in the "Web way", that is without central authority, with few basic rules, in a scalable, adaptable, and extensible manner.

1 Introduction

During the last two decades there has been an impressive progress in the fields of automation of formal reasoning and mechanisation of mathematics. Still, all the available tools, whose development typically started in a pre-Web era, clearly suffer by the adoption of an old application-oriented architectural design, and obsolete technological solutions which hinder their evolution and, especially, their integration with the World Wide Web.

The main point of the paper, and of the HELM Project, is to stress the potentialities offered by the eXtensible Markup Language (XML) and most of its related technologies, for the the creation and maintenance of a virtual, distributed, hypertextual library of formal mathematical knowledge. The crucial point is the evolution from the old application-oriented management of information, to a new content-centric architectural design [10], enabling the exploitation of information in the "Web way", that is without central authority, with few basic rules, in a scalable, adaptable, and extensible manner [14]. Establishing a layer of simply accessible and universally understandable data is the key to allow the definition of sophisticated search engines and interoperable services, and to enable higher degree of automation and more intelligent applications.

The eXtensible Markup Language does not only provide a central technology for storing, retrieving and processing mathematical documents, but naturally brings major benefits in all the following, crucial issues:

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

Standardisation Having a common, application independent, 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.

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 & 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.

Modularity The "XML-ization" process should naturally lead to a substantial simplification and re-organisation 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.

All these points will be discussed at length in the following sections.

At the best of our knowledge, the only other significant Projects in the world focused on the application of XML technology to the realm of mathematics have been the W3C Working Group on MathML (`http://www.w3.org/Math/`), of which we are a member, OpenMath (`http://www.openmath.org`), and the MathWeb Project in Saarbrücken (`http://www.mathweb.org`) which recently produced an extension to OpenMath called OMDoc[28, 29, 30]. MathML is an essential component of HELM (see section 2.6). The relations between HELM and OMDoc will be detailed in the following sections. As for OpenMath, the main difference with respect to HELM is that the former has been mainly focused on computer algebra systems and interoperability issues; in particular *proofs* play a pretty marginal role in OpenMath. On the other side, we consider proofs as a main component of mathematics (maybe the most interesting part of it); as a consequence

1. HELM is more naturally oriented towards Proof Assistant Applications [24, 25] than Computer Algebra Systems. In particular, we used the Coq [15] Proof Assistant of INRIA (France) as a paradigmatic example of this kind of applications[1]. Coq is a well-maintained and pretty complex system, based on a very rich, higher-order logical framework called the Calculus of (Co)Inductive Constructions (CIC).

2. the main emphasis of HELM is on rendering, browsing, management, searching and retrieving issues. The improvements in interoperability are just a welcome side-effect of the overall methodology.

We assume the reader is familiar with the definition of XML and some of the main W3C recommendations.

2 Structure of mathematical knowledge

Mathematics is a richly structured language. In this section we discuss the main categories of linguistic elements appearing in mathematical developments and the related issues. The discussion is mostly oriented to an automatic elaboration of mathematical knowledge.

2.1 Names

In the practice of mathematics, the choice of names is essential, in that names bring a useful and important meaning, often fixed by an ancient mathematical tradition. In general, names

[1] See also [17] which is a related work aimed at exporting Coq developments into OMDoc.

should be short and meaningful; moreover the same name may have different meanings in different mathematical contexts: a long name, or path, may be used to overcome the ambiguity in this situations, naturally leading to consider structured names, composed by specifying a prefix denoting a path in some hierarchical structure. In the terminology of the Web, each referentiable (mathematical) entity should have a unique Universal Resource Identifiers (URI) [16] denoting it. A generic URI is made of a formatted (structured) string of characters whose intended meaning is associated by the applications managing it[2]. 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`".

2.2 Expressions and Propositions

This is probably the best settled and understood part of the mathematical language. Most systems for automatic elaboration of mathematics rely on similar, first-order encodings of expressions and proposition, which is particularly important for interoperability issues. Recent XML proposals, such as MathML-content [3], or the Object level of OpenMath (which are essentially isomorphic), provide a good ground for standardisation in this field. In particular, the base set of MathML-content elements is meant to be adequate for simple coding of most of the formulas used from kindergarten to the end of high school and the first two years of college, that is up to A-Level or Baccalaureate level in Europe. Subject areas covered to some extent in MathML are: arithmetic, algebra, logic and relations, calculus and vector calculus, set theory, sequences and series, elementary classical functions, statistics, and linear algebra. Content markup consists of about 100 elements accepting a dozen attributes. The majority of these elements are empty elements corresponding to a wide variety of common mathematical operators, relations and functions, such as `partialdiff` (partial differentiation), `leq` (less or equal) and `tan` (tangent). Others elements, such as `matrix` and `set`, are used to encode various mathematical data types, and a third, important category of content elements such as `apply` are used to apply operations to expressions and also to make new mathematical objects from others. MathML 2.0 has been conceived as an extensible language: the constructor for defining new user-defined operators is the `csymbol` element, largely used inside HELM.

2.3 Proofs

Proofs are an essential aspect of mathematics. The mathematical investigation of proofs is a recent branch of Logic which goes under the name of *Proof Theory* (see e.g. [32, 35, 33, 31, 21]). According to this theory, and the so called "Curry-Howard" analogy ([23]),

[2] An important difference between HELM and OMDoc is that we use structured URIs, while OMDoc relies on flat ones. Structured URIs help their mining, facilitate search and retrieving operation, and naturally induce a hierarchical organisation of metadata, opening the way to simple inheritance mechanisms.

formal proofs are just another category of mathematical *expressions* and can be conveniently represented and serialised by standard means.

More complex is the issue of recovering a "natural" presentation of proofs from their formal encoding. Two orthogonal and compatible ways seem to be open here (and both are currently investigated in HELM):

1. automatic reconstruction into natural language (see e.g. [19, 20, 17])
2. computer assisted annotation, in the spirit of the Annotea project [26].

Both approaches heavily rely on the use of *redundant information* (see section 2.7) to improve the readability of the text. An opposite problem is that of avoiding to clutter the proof with too many formal details, and microscopic steps (that typically afflicts formal developments and any kind of computer understandable information). The natural solution is that of allowing the user to inspect the proof at different levels of detail, possibly "exploding" subproofs on demand (this approach is currently under investigation in HELM). The main difficulty of subproof explosion is recognizing in the term the reasoning threads, in order to show or hide atomically whole threads.

2.4 Theorems and Definitions

Theorems and Definitions are the "basic blocks" of mathematical developments (they correspond to the minimal explicitly referentiable entities). Accordingly, this is also the granularity of information in HELM: each definition/statement is a stand-alone XML-file, uniquely identified via a URI (inner information is still accessible by means of XPath technology; actually, to improve computational performance each element of HELM documents has a unique ID-attribute inside the document). Although this solution may look natural, it is not customary of Proof Assistant applications, where information is usually saved in bigger clusters (theories, or sections). The latter organisation has major drawbacks for the maintenability of the library:

1. we cannot access or use a result without requiring at the same time the whole theory inside which it is defined. Since this "theories" are often very big, many times authors redefine locally the required results, leading to a useless and confusing duplication of information.
2. extending a theory or a section requires the recompilation of the whole section. As a consequence, the "library", which is the result of the contributions of many different authors which do not have any access to contributions of other people, tends to have a "flat" and disorganised structure, which hinder its development as a joint and cooperative effort.

2.5 Documents and Theories

The distinction between a document and a theory is not so evident. Roughly, a "document" is a more or less arbitrary collection of Definition/Theorems, suitably assembled by some author for presentational purposes. A "theory" is a a well organised sets of Definition/Theorems, typically respecting their logical dependencies, and possibly supporting complex specification mechanisms, like inclusion, coercion and inheritance; they are traditionally called "modules", in the proof assistant and functional communities. The distinction is obviously blurred, since each document could and should profit by the sophisticated mechanisms supported by the theory model. A possible way to make the distinction more clear is at the level of Uniform Resource Identifiers: the structure of URIs should reflect the logical organisation inside a given *theory*, but they should be *freely* referentiable inside any *document* (that does not prevent the possibility of automatic checking). Our current effort inside HELM has been mostly oriented towards "documents". OMDoc [29] is a first standardisation attempt of a strong notion of "theory". The main problem with OMDoc is that the complex issue of modules is still an open research field in the proof assistant community: each application is likely to develop in the near future its own module system, possibly integrated inside the logical kernel, and these solutions may be easily incompatible with the OMDoc specification. Maybe some standardisation is possible indeed, but this should be the result of a complex negotiation and cooperation between many different parties. At present, any attempt to impose a specific solution is very likely to be rejected by the scientific community. Nevertheless, OMDoc has the important merit to provide an interesting and constructive starting point, emphasising a major problem of current mathematical developments.

2.6 Notation

The relation between meaning and notation is complex, and part of the descriptive power of mathematics surely derives from its ability to represent and manipulate ideas in a highly evolved system of two-dimensional symbolic notations [27, 22, 34]. Modern mathematical notation is the product of centuries of refinement, and this knowledge is a main component of every mathematical development. Moreover, mathematical notations are constantly evolving along with the progress of the discipline, requiring sophisticated and highly modular means to relate notation to content. The final aim is to be able to change notation with the same confidence and easiness we change a symbolic font in modern authoring languages. To this aim we need:

- a good presentational language. MathML-presentation [3] is indubitably a major breakthrough for Web-publishing issues. It provides a sophisticated editing environment, consisting of about 30 elements which accept over 50 attributes. Most of the elements correspond to layout schemata, which contain other presentation elements. Each layout schema corresponds to a two-dimensional notational device, such as a superscript or subscript, fraction or table. In addition, there are the presentation token elements `mi`, `mn` and `mo` that respectively stands for identifiers, numerical constants and operators, as well as several other less commonly used token elements. The remaining few presentation elements are empty elements, and are used mostly in connection with alignment.

  Unfortunately, there are no satisfactory implementations available yet (Amaya and Mozilla support only a subset of MathML, and their performances are quite low). Moreover, the browser should support sophisticated forms of interactions (e.g editing of sub-expressions) which are potentially enabled by MathML but never explored so far. For this reasons, we developed inside HELM a brand new engine, named GtkMathView (`http://www.CS.UniBO.it/helm/mml-widget`), with rendering[3] and interaction capabilities for documents embedding MathML presentation markup. The widget will soon adopt Gdome2 [18], a new DOM level 2 Gnome implementation that is another by-product of HELM. Thanks to Gdome2, it will be possible to easily integrate it with other markup engines, with the final aim of developing an architecture in which different kind of markup could be freely intermixed in the same document and rendered by cooperating widgets.

- a *standard* mean to associate notation to content. In [13] we advocated the use of XSL-Transformations (stylesheets), [9] to this aim. XSLT is a simple, rule-based, declarative language, explicitly conceived as a mean to specify the *styling* of an XML document by

[3]Both to screen and Postscript.

transforming the specific XML-dialect of the input document into a formatting language suitable for rendering issues (HTML, Formatting Objects, or whatever). 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 [8] language, that allows to match elements according to their values, structure and position in the source tree. XSLT is very simple, easily maintenable, and easily extendible. Moreover, most of the stylesheets (especially the notational ones) have a simple and repetitive structure, offering the possibility of automatically generate them starting from an abstract and concise representation of notational information.

2.7 Redundant Information

As any language especially meant for human readability, mathematics is highly redundant. This is especially evident at the level of proofs, which is the most discursive part of the mathematical language. For instance, when inside a proof we meet a statement of the kind "by applying theorem x to the hypothesis y and z we obtain P", the intermediate conclusion "P" is completely redundant, since it may be inferred by x, y and z. Its only (but essential!) purpose is to improve readability. In the rest of the paper, we shall refer to this kind on information as the "Inner Types" of proofs, since in a Curry-Howard correspondence they correspond to the types of inner nodes in the abstract syntax tree of the proof.

From the point of view of automatic elaboration, redundant information is just a burden: either it could be simply ignored by the application (with loss of space and parsing time), or it imposes a lot of additional consistency checks. On the other side, it may be essential for different forms of elaboration, such as rendering. For these reasons, it is important to have an explicit representation of all useful "redundant" information, but it is equally important to keep it apart from the "core" information, which is the real informative content (the minimal information required for automatic checking).

2.8 Metadata

To enable and facilitate specific functionalities such as searching and indexing we need to associate to documents some additional information, describing properties of the document and relationships among entities referred to by documents. The Web terminology for this kind of information is *metadata*, that is *data about data*, and typically comprise labelling, cataloguing, and descriptive information of several kinds. Some metadata are user-specified (author, editors, keywords, and so on). Other kinds of metadata may be automatically generated from the document itself, as, for example, the set of documents that refers a given one; in this case, metadata are used to data-mine in a batch process redundant information too expensive to be computed on-the-fly.

The W3C Resource Description Framework (RDF) [5, 6], provides a general model for representing metadata as well as a syntax for encoding and exchanging these metadata over the Web. This standard approach is domain neutral, so it doesn't make any assumption about an application domain, and it provides interoperability of independently developed Web servers and clients, and in general between applications that exchange machine-understandable information on the Web: documents described by RDF metadata are effectively indexed by search engines.

Metadata management in HELM makes a major difference w.r.t OMDoc. In the latter specification, metadata are mixed together with other levels of markup inside a single DTD specification. Indeed, merging together in the same document different information levels (content, rendering, metadata) to improve one kind of processing, leads to the risk of making less effective all the others. For instance, this is what already happened with SGML in the publishing world: documents became loaded with many different levels of markup (for macro- and micro-structuring, making context-dependent semantic distinctions, etc.) thereby making the resultant collections difficult to work with.

On the contrary, in HELM we adopted the general approach of layering levels of processing, in order to manage complexity. As a consequence, content, presentation, redundant information and metadata are all kept separated from each other.

3 Layers of mathematical representation

In relation to any tool for the mechanisation of mathematics, it is possible to identify at least four different representation layers of mathematical information.

internal encoding The first layer is the way information is internally encoded inside the specific tool. This description is eventually application-dependent. Note also that it is a really *formal* description, at least in the somewhat crude sense that it must be machine-understandable.

logical encoding Good tools usually have a clear semantics, based on a clean and compact logical system. The internal encoding has thus a natural logical interpretation that provides a purely logical description of the information. This description is still formal, and we can usually go back and forth from the previous descriptive layer to this one with no loss of information. The important point is that this description is not any more application-dependent: it just depends form the specific logical system the application is using.

content description This layer is meant to encode the "actual content" of the information, regardless of the grammatical (and semantical) details of the foundational logical system. To make an example, we understand the meaning of the integer constant 0 independently by its specific logical encoding or by the axiomatisation of integers inside the logical system. This level is thus meant to supply an "abstract syntax", which can be particularly useful for translation purposes. The syntax is supposed to cover all the symbols typically used to represent concepts arising in a particular area of mathematics. The intended semantics of these symbols is supposed to be fixed by common agreement (as in MathML) or via auxiliary documentation. For instance, the OpenMath Project uses Content Dictionaries (CDs) for this purpose. CDs are public documents, representing the actual common knowledge among OpenMath applications, fixing the "meaning" of objects independently of the application. The application receiving the object may then recognise whether or not, according to the semantics of the symbols defined in the Content Dictionaries, the object can be transformed to the corresponding internal representation used by the application.

presentational layer The final layer is the purely presentational layer. An important part of the descriptive power of mathematics derives from its ability to represent formal concepts in a highly evolved, two-dimensional system of symbolic notations. We must be eventually able to recover this rich presentational format, by a process of re-mathematization of formal content [13]. In a broader sense, under the presentational layer we also consider all the functionalities offered to the end-user in order to facilitate browsing, searching, retrieving and management of the mathematical repository.

3.1 The current state

Most of the current tools for the mechanisation of mathematics, just rely on two of the previous layers: the internal and the presentational one (and often, the "presentational" description is very poor). In Proof Assistant applications, proofs are usually saved in two format: a script of tactics, which is essentially the sequence of commands issued by the user to prove a statement during an interactive session, 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. In particular, the language of tactics is really system dependent, often partly documented, in continuous evolution. Moreover, we can not even speak of the language of tactics as a high level language that is "compiled" to the language of terms: even if writing a script interactively is much simpler than writing the term by hands, once written it becomes impossible to understand it without replaying it interactively. The deep reason is that the semantic of tactics is not compositional, but depends on the current subgoal and its environment.

The notational support provided by user-friendly interfaces has been typically tightly integrated with the systems, as a set of parsing and pretty-printing rules between the presentational layer and the internal encoding. This approach lowers the modularity of the architecture, and essentially prevents sharing of results and reusability of software components.

3.2 The HELM approach

One of the first methodological assumption of HELM has been to exploit all the previous layers. XML can be naturally adopted as a neutral, application independent meta-language for the logical layer. Each logical system will need its own specific DTD, since we cannot escape, both theoretically and practically, the multilingual environment of the foundations of mathematics. But this is not a limitation: the *standardisation* we are pursuing is not at the *logical* level, but at the *technological* one. By explicitly introducing the logical layer, we already made a major step, decoupling the information from the application which is meant to elaborate it. By using a common metalanguage like XML for the description of this layer, we open the possibility to use standard commercial tools for further processing, augmenting at the same time the modularity and interoperability of software components. In particular, all further transformations towards content and presentational descriptions can be suitably defined in terms of XSL-transformations (stylesheets) between XML documents.

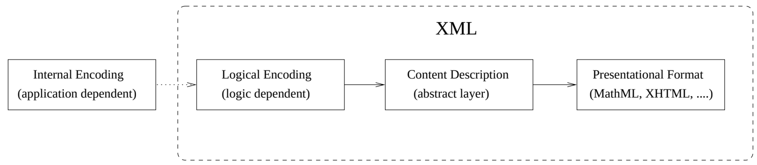

Figure 1: Representation layers of mathematical knowledge.

3.3 HELM vs. OpenMath

OpenMath makes no distinction between what we called the logical and the content level. The reason is twofold. First, their intended applications (computer algebra systems) are sufficiently uniform at the logical level (typically, first order classical logic) to avoid the need for a specific intermediate format. Second, the OpenMath focus is primarily on mathematical statements, neglecting proofs which are typically much more system dependent.

In OpenMath, the conversion of a content object to/from its internal representation in a software application is performed by an interface program called Phrasebook. The translation is governed by the Content Dictionaries and the specifics of the application. It is envisioned that a software application dealing with a specific area of mathematics declares which Content Dictionaries it understands. Since Phrasebooks are an important component of the general architecture of OpenMath, they should be public and subject to some kind of standardisation. Adding our logical layer, Phrasebooks could be conceivably written by means of XSL-transformation, with a major improvement for the modularity of the whole architecture.

It is finally worth to remark that OpenMath's Content Dictionaries are not "first class citizens". They are machine-readable, but not machine understandable; their content is largely informal (the only part which is fully formalised is administrative information like review and expire dates) preventing the possibility of any automatic elaboration of their content (such as, for example, consistency checks). They are essentially conceived as background references for the implementors of Phrase-books.

4 Interoperability

Suppose to have two systems, addressing the issue of their interoperability. The problem is to pass from an internal encoding in system A to an internal encoding in system B and vice-versa. As a consequence you need expertise on both systems, requiring a tight cooperation within the two developer teams. If the information is exported in a clean logical representation, the problem is essentially reduced to a clean logical problem of mutual encoding of two formalisms. If moreover both logical representations relies on a common and widely used meta-language like XML, the actual implementation of the mutual encoding is surely easier, and may profit of standard technologies. In conclusion, anybody is free to make its own interoperability experiments, without requiring *any* knowledge at all of the two systems.

Let us consider a simple example. In order to check that we exported all relevant information from Coq we wrote our own proof-checker for the Calculus of Inductive Construction. Having an external type-checker is a very interesting and general problem since there is no reason to rely on Coq to check Coq proofs. Developing an external proof-checker is a clear example of interoperability between different systems. The goal is to rewrite only the kernel of the proof-assistant, that works on proof objects. As a consequence we are not interested in script files, since we do not want to rewrite also the interpreter for tactics. Hence, we need to "export" proof objects in some intermediate serialisation format, and XML provides the standard solution.

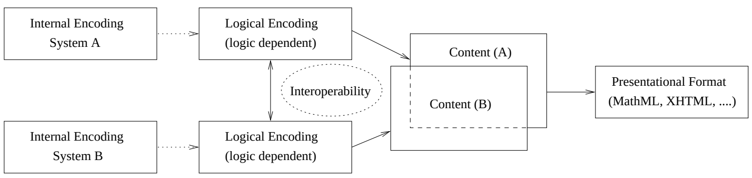

Figure 2: Interoperability in HELM.

A more sophisticated form of interoperability can be also exploited at content level. The idea is that the content level of different systems should largely overlap (if the two systems are based on a common logical systems, and once a common abstract notation for usual mathematical signatures has been fixed, the two content level essentially coincides). If the content level is sufficiently formal to allow a backward translation into different logical encoding, we have already achieved a full interoperability between the two systems. In any case, we have clearly identified a *region* between the logical and the content level where interoperability should be exploited. Any interoperability issue which could be entirely solved at content level is then an enhancement towards full interoperability. At present, it looks feasible to make two different systems understand each other *statements* by a natural encoding into a common content representation; it looks much more difficult to make the two system understand each other *proofs*, for the simple reason that an "abstract" syntax for proofs is much less evident than for terms and propositions.

5 Publishing

In Figure 3 you may see a screen shot of the Web interface of HELM in which mathematical formulas are rendered in HTML. We can also output MathML presentation markup, that can be rendered with the tools described in section 5.2.

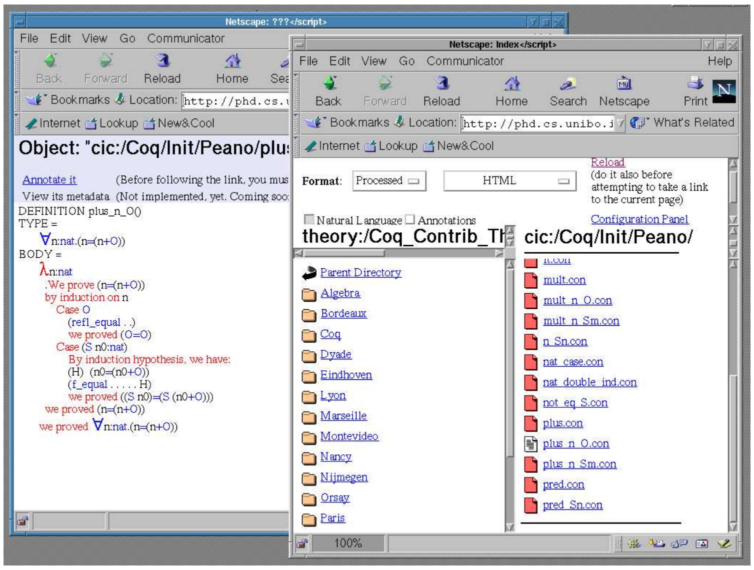

Figure 3: HELM HTML-interface

In Helm there are two main ways for browsing the library, that is via Theories (or documents, see the discussion in section 2.5) or via the logical hierarchy of Definition, Theorems, etc., providing a direct access to individual Objects. This is reflected in the organisation of the interface (the top window in Figure 3): the hierarchy of Theories appears on the left, while the hierarchy of objects is on the right. A Theory in HELM is an arbitrary (structured) collection of mathematical Objects, suitably assembled by some author for presentational purposes. Definition, Theorems, and so on may be intermixed by explanatory text or figures. Inside a Theory we only visualise statements without proofs: a link to the corresponding proof objects allows the user to inspect proofs, if desired.

The HTML presentational markup interpreted by the browser is generated *on the fly* starting from a low level XML description of the corresponding file. This requires a complex transformation of the document; all the transformations are performed by means of XSL-stylesheets, as detailed in the next section.

We offers the choice between several different modes for accessing the information, which can be easily selected via buttons and menus in the top bar:

Raw This returns the low-level XML encoding of the information. There are three sub-modes:

- **CIC** pure objects of the Calculus of Inductive Constructions.
- **Types** Inner Types of the object.
- **Annotation** textual annotations of the object.

Moreover, you may switch on/off automatic decompression, and the resolution of URIs to URLs inside the document.

Processed Again there are several sub-modes, according to different forms of processing:

- **HTML** HTML output format interpreted by the browser.
- **MathML-content** intermediate "abstract" encoding of the information.
- **MathML-presentation** if you dispose of a MathML compliant browser, you may use this mode as a valid alternative to HTML.

In this case, you may also switch on/off the transformations in charge of rendering formal proofs into a more friendly and natural-language like presentational format, and/or the transformations interpreting possible annotations of the proof.

This complex mixing of possibilities requires both a tool for managing sequences of cascading applications of XSLT stylesheets, and some dynamic mechanism to resolve a link to a suitable HTTP-request parametrised according to the user preferences (we use JavaScript, to this aim). All these aspects will be discussed in section 7.

5.1 Transformations

In HELM, the transformation of a document from the internal representation in some logical environment to its final rendering essentially passes through four phases: exportation, transformation, presentation, and rendering. Figure 4 provides the overall, simplified architecture of these phases. Rendering will be discussed in the next section, together with the tools required for the management of stylesheets. In this section we focus on the first three phases.

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

In Helm, the only *persistent* level of the library is the low-level XML-encoding of the information obtained after the first phase: all other formats are generated *on the fly* by means of XSL-transformations. Most of these transformation are pretty complex: we heavily rely on the inclusion mechanism of XSLT to organise stylesheets in a coherent and easily maintenable structure. For instance, Figure 4 describes the inclusion hierarchy for rootcontent.xsl (the stylesheet responsible for transforming low-level proof-objects into their intermediate representation).

Finally, as it is clearly evinced in Figure 5, there are two main flows of transformations, according to the two possible outputs, namely theories or individual objects. The most complex transformation process obviously concerns objects (and its sublevels: namely propositions, terms and proofs, comprising the required notational support). Theories, are essentially structured wrappers for objects, and do not require major transformations.

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

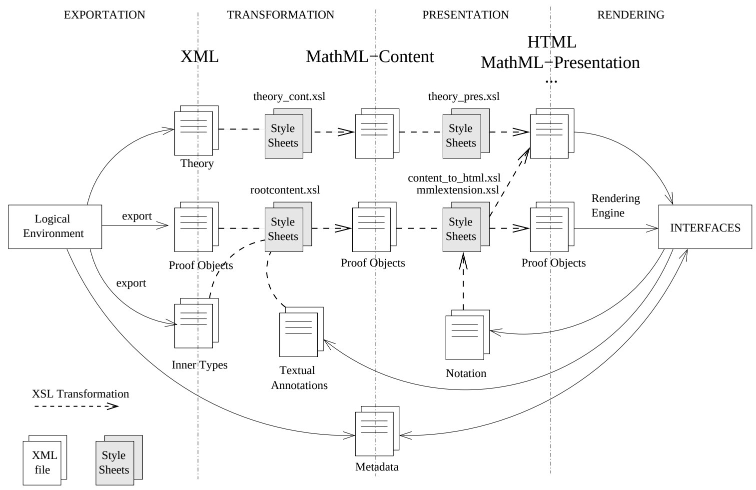

Figure 4: Helm Transformation Phases and main stylesheets

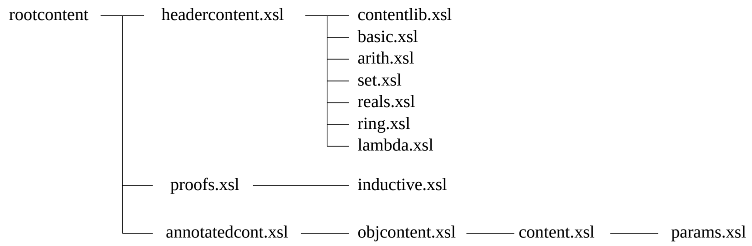

Figure 5: Hierarchical organisation of rootcontent.xsl

5.1.1 Exportation

In the first phase the document is exported into a suitable low-level XML-dialect which is specific to the particular logical system used by the given application. This is a batch process, producing the actual, persistent library of HELM.

The necessity of having a specific description is motivated by the fact that the information encoded is very different both in content and format from one system to another. A general purpose language would hardly reflect all the small but essential details of the specific internal representation of the information inside a given logical framework.
The main problems of this phase are to decide which information is worth exporting, to choose the right granularity of XML documents, and the actual definition of the Document Type Descriptor.

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

An important choice of HELM has been to decouple the information according to its meaning and usage. The basic information is the logical term. Inner Types are a redundant information which is worth to keep explicitly, but should not confused with the "core" information. For this reason they are kept in a separate file; each type refers to a given subterm by means of the unique ID attribute of the root node of the subterm. The same mechanism is used for manual annotations of proofs: each annotation refers to a specific fragment of the proof identified by means of its ID. Once more, metadata for each proof-object (or theory document) are kept aside from it, in a companion document. This architectural choice makes a substantial difference with respect to OMDoc, where all different markup is essentially mixed together inside a single document.

5.1.2 Transformation

The second phase is the core of the transformation process. In this phase the document is transformed, by means of stylesheets, into a suitable intermediate "abstract" representation (a pointer to the formal content is preserved as an Xlink). This intermediate level is meant to improve the modularity of the whole architecture. Many different formal notions, from the same or even different logical environments, are typically mapped here into a same intermediate notion. Think for instance of the definition of equality, or of an order relation: their formal definition may be very different from one system to another (or from a subtheory to another), but their intended presentation (and intuitive meaning) is the same. So, there is no point in defining a specific presentation for each formal notion. We just define presentation for the intermediate content level, mapping all formal notions into the corresponding "abstract" one, with the intended representation. In HELM, we have adopted MathML-content for the intermediate representation of formulae and proofs, and have defined a new markup for the theory and metadata level.

During the transformation phase we must also heavily work on proofs in order to put them in a form more suitable to human reading. Typically, this requires a major reorganisation of the structure of the proof (see e.g. [19, 20]): in proof assistant, proofs are typically generated in a bottom-up fashion, while we naturally expect a top-down presentation, where subterms (sub-proofs) appears before conclusions. Another complex issue is that of recognising and managing induction principles, (one of the main proof-mechanism of constructive mathematics and Proof assistant applications). During this phase, proofs are also integrated with their Inner Types (intermediate conclusions, see section 2.7). One of the most interesting achievements of HELM has been to prove that even these complex transformations can be feasibly performed by means of XSL-Transformations. A different approach is the one of Caprotti, Geuvers and Oostdijk [17] that generates the natural language rendering off-line, using a tool written in Java; the output is then included once and for all inside the mathematical document (that is an OMDoc instance). Even if the initial implementative effort is surely greater, our approach has clear advantages in terms of user-configurability: changing the natural language output, for example associating a particular verbalization to an operator or changing the whole output to another language, just amounts to adding or overriding a few stylesheet templates. Moreover, this can be done on-the-fly on a per-user basis.

5.1.3 Presentation

In the third phase the document is transformed from its intermediate, abstract representation to the final presentation format (currently we produce either MathML-presentation or XHTML; other languages could be exploited in the future). This final transformation is based on a bunch of pre-defined or user-defined stylesheet, containing notational and stylistic intelligence. The transformation from MathML-content to MathML-presentation requires less than 2000 lines of XSLT (suitably organised in a hierarchical structure[4] similar to that

[4]We use, among others, a stylesheet, compliant with the last specification of MathML, written by Igor Rodionov, of the Computer Science Department of the University of Western Ontario, London, Canada.

of Figure 4); the extension to new `csymbols` external to the base set of MathML-content requires for 1400 additional lines (similarly for XHTML). Most of these stylesheets have a very simple and repetitive structure and we are currently studying the possibility to automatically generate them from a more abstract and concise representation of notational and stylistic information. For example, most of the rules dealing with operators of the same arity are similar, and could be inferred from the arity, the associativity, the type (infix/prefix/postfix) and the precedence index of the operator.

5.2 Rendering and HELM interfaces

We want every user to be able both to consult and to contribute to the library requiring as few client-side software as possible. In particular, at least for simple consulting, we propose a Web interface requiring only a common browser. A major technical problem we have to face, though, is that no already available browser fully implements the MathML specification and behaves correctly on our quite peculiar documents (for dimensions and level of table nesting). Moreover, hyperlinks could be added to MathML documents only via XLink and, as a consequence, we require browsers both MathML and XLink compliant. Finally, even if we can expect such browsers to be developed in a few months, it is quite unlikely that we will soon have the possibility of freely nesting different kinds of markup (e.g. XHTML and MathML) in the same document; note that this feature is needed to render both mathematical documents and user-annotated formal proofs.

For the previous reason, we have developed both stylesheets to produce MathML and a reasonable approximation in XHTML relying on the widespread "*font*" symbol. The main issues that can not be easily addressed in XHTML are:

1. **Layouting of oversized formulas.** A MathML engine has enough information to correctly break lines that exceed the page width, re-indenting the output accordingly. For XHTML, we simply let the browser create an oversized canvas. Note, however, that in our stylesheets we already do a good work computing a coarse-grained layout so that the event of oversized formulas is very rare and not very severe.

2. **Multiple-depth formula rendering.** A MathML presentation element, named `maction`, is used to create a node having many children of which only one is shown; the user can switch the visible one. We are going to exploit this element to allow to browse the proof as a collapsing tree, where the user is free to expand the proofs in a progressive way, augmenting the level of detail only locally. This can not be reproduced in XHTML. To achieve a similar effect we can simply render again with different parameters the whole proof. Waiting for the new rendering, though, will probably annoy the user, especially for huge proofs whose rendering requires many seconds.

3. **Smart selections.** A MathML engine could easily give the user the possibility to select sub-expressions in a smart way. Even if it is unlikely that standard browsers could be exploited for complex interactions (e.g. editing), just clearly identifying sub-expressions through selection is sometimes very helpful to understand complex terms.

As a compromise to cope both with the previous limitations of XHTML and the unavailability of MathML compliant browsers, we have built a plug-out for Netscape for Linux to render MathML. The plug out, that is based on GtkMathView, remotely controls Netscape in such a way that the browser and the plug-out windows are kept in synch. This means that the XHTML page always refers to the object shown in the plug-out; following an hyperlink in the plug-out or pushing the back button of the browser updates both documents. That is important because the XHTML page holds a control frame that proposes to the user the actions allowed on the current object (e.g. type-check it).

In this way, the control frame and its JavaScript logic have not to be reproduced in our plug-out, that is kept extremely simple. This solution, though, has some limitations too:

1. It requires the user to install client-side software. Moreover, it works only for Linux boxes and, without modifications, only with Netscape Navigator.

2. Due to the way the plug-out is kept in synch with Netscape, progressive rendering is not allowed. Hence, you have to wait for all the file to be downloaded for Netscape to pass it to the plug-out for rendering.

3. The plug-out solution does not work for MathML embedded inside other kinds of markups.

If we leave the problem of simply consulting the library and we focus on more advanced forms of interaction, for example annotating formal proofs with informal descriptions of some of the proof steps, the only feasible solution seems to require the user to install client-side software. Here we are implicitly assuming that downloading an applet every time we need it is too time-consuming. The reason is that the applet should be at least able to render MathML and this requires a huge amount of bytecode to be downloaded even for simple examples. Despite these limitations, we still would like to minimise code replication and to keep as much as possible a coherent and unique interface. Hence, we are going to develop other plug-outs to be integrated in the Web interface. For example, we already have a plug-out to annotate proofs that can be invoked on the proof now displayed from the control frame of the interface. To select the wanted plug-out, a different MIME-type is returned by the processor.

To sum up, an on-line version of the standard library of Coq V7 can be found at the address `http://phd.cs.unibo.it/helm/library`. A plug-out to render MathML documents and another one to annotate proofs can be freely downloaded. Soon we are going to integrate in the Web interface also our proof-checker, that will run on the server-side.

A detailed description of the implementation will be given in section 7. Before it, though, we still have to describe the distribution model for HELM library.

6 The model of distribution

The main requirement for the user interface of HELM was to keep a minimum burden on the user: no client-side software is needed to consult the library and as few as possible to interact with it. Now we want the same thing to happen for contributing[5] to the library. In particular, we want any user with a Web space (either HTTP or FTP) to be able to contribute a document/object to the library without having to install at all any particular software. The main reason is that often the Web space of a user is hosted by a provider that does not allow new software to be run. Moreover, the simple HTTP publishing model has already proved itself really effective in creating really distributed hypertextual libraries of knowledge. This is a motivation against a more centralised solution, such as a net of cooperating databases to which a user can submit a contribution. Note that this is the solution adopted in MathWeb.

Our distribution model has been also designed to exploit the peculiar property of formal mathematical documents of being immutable. The reason for immutability is that 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). This is exactly what happens with packages of operating systems distributions that, once released, can be modified only changing their version numbers.

[5] In this context, contributing means making available a development to others; it does not mean creating a new object or document.

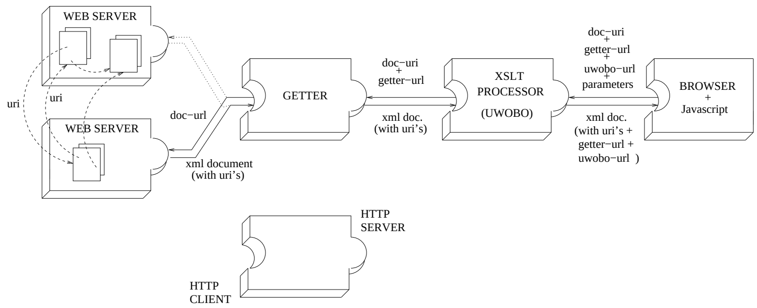

Figure 6: Implementation components diagram.

Because documents are immutable and because we already identify them with logical names (URIs) instead of physical names (URLs), many copies of them could be simultaneously present on different servers and a user is free to get them from the nearest or less overloaded server, achieving load balancing. How this is implemented in describe in the next section.

Finally, users cannot be forced to retain a copy of their documents forever, even if other documents refer to them. Allowing different copies on different servers, though, we give the possibility to users interested on a contribution to make a local copy of it and to start to distribute it. In this way, interesting documents augment the number of their instances, avoiding the danger of disappearing, while uninteresting ones could simply stop wasting space, being deleted.

Three main problems are still unaddressed. The first one is the choice of a naming policy to avoid users working independently to choose the same URI for different documents, creating name clashes. Up to now, we have not chosen or implemented a naming policy yet. To face the issue, one solution is a centralised naming authority, even if other more distributed scenarios may be considered.

The other two problems are related together. The first one is the way a user can locate and download (an instance of) a document given only its name and a list of servers possibly providing it. The second one is where to elaborate the documents, e.g. applying the transformations described in section 5.1. In fact, we have chosen to avoid any additional software both on *distribution sites*, which are the HTTP or FTP servers holding users contribution, and on the client-side, which is the host of the user consulting the library. Both problems are addressed in the next section.

7 Implementation

The HELM architecture requires at least three components, which are distribution sites, standard browsers and plug-outs, and active components, such as XSLT processors, to elaborate the information. Distribution sites are simply HTTP and FTP servers, widespread all over the world; user browsers are HTTP clients and run on the user host. We do not want to require all the other components to run on a a particular host.

Because they must provide answers to browsers, they must provide an HTTP server interface; because they must ask data to distribution sites, they must also be HTTP clients. Hence, we have chosen to organise the whole HELM architecture as an HTTP pipeline. Every node of the pipeline can be seen as an object providing different methods, each node taking in input a list of arguments. A different URL is associated to every method, with the search part of the URL (the one after the question mark) used to pass the actual arguments in the standard way. Figure 6 shows the pipeline to apply XSLT transformations to document. The pipelines for other tasks (e.g. proof-checking) are obtained simply substituting the XSLT processor component.

The module client of the distribution sites is the *g*etter, which maps URIs to URLs and hence documents. The whole module has been largely inspired by the APT packet management system (`http://www.debian.org`). The main method of the getter takes an URI and returns a copy of an instance of the document identified by the URI, downloaded from a distribution site providing it. In order to know which documents a server provides, each server publishes a list of the URIs of its documents with associated the respective URLs. Users of the getter, instead, provide to it an ordered list of servers. On a regular basis, the getter contacts every server on the list and retrieves the list of documents provided with the associated URLs, building a local table (a NDBM database) mapping each URI to the URL of the first server of the list providing a copy of the document.

Due to the high verbosity of XML files, we give the user the possibility to store files on the distribution servers in a compressed form. The getter is also responsible to deflate them before giving them back to its client. Moreover, the getter is able to retrieve the files both from HTTP, FTP and NFS servers, giving them back to the client using a uniform HTTP server interface. Finally, the getter is supposed to reside closer to the user than the distribution server; hence, our getter also implements a cache, reducing downloading time of already retrieved documents. All these reasons forced us not to use the HTTP redirect method to map URIs to URLs, as done by PURL (`http://www.purl.org`) which is an otherwise similar tool to resolve persistent URLs (URIs, in fact) to URLs[6].

The actual implementation of the getter is in PERL, the choice of the language being motivated by the simplicity of the task, the portability of the language and the availability of already developed libraries implementing HTTP servers and clients, files compression and NDBM management.

Another important component of HELM architecture is UWOBO (`http://www.cs.unibo.it/helm/uwobo`). UWOBO is an XSLT stylesheet manager, implemented in Java and based on Xalan (`http://xml.apache.org`), whose main method is used to apply a list of stylesheets (each one with the respective list of parameters) to a document. The stylesheets are pre-compiled to improve performance. Both stylesheets and the document are identified using HTTP URLs and can reside on any host. In particular, the document URL provided in HELM is usually the invocation of the getter method to download a document whose URI is also given in the dynamic part of the URL.

Let's now describe a typical use case of the HELM interface. First of all, as soon as a user enters the HELM site, he is asked to specify the URLs of the getter and UWOBO to use. These informations, that can be changed at any moment, are never forgot, being passed back and forth as parameters to the server each time a new page is requested. Then, a list of known documents (both theories and objects) is presented. To this aim, a special "index" method is used to ask the getter the list of all the URIs of the known documents (that obviously depend on the list of the known distribution servers). This is an XML document which is then processed by UWOBO to transform it into an HTML page, with some JavaScript code whose functionalities we are soon going to describe. The user now selects from the interface the document to look at, together with some rendering options, such as the expected output format or the notation to apply. Hence, the JavaScript code maps the rendering options to a list of stylesheets and stylesheets parameters and assembles them into a suitable URL. This URL is quite complex: essentially, it is a request to UWOBO to process a document with a given sequence of stylesheets and parameters; however, since UWOBO retrieve the document via the getter, the URL of the document to retrieve is in turn a HTTP-request to the getter

[6]Being PURL general-purpose, it can not rely on document immutability. Hence, at most one copy of a document could be available and no load balancing is provided.

(with the proper method and parameters). Typically, more complex is the pipeline, longer is the actual URL (and the pipeline we described is already a simplification of the current pipeline of HELM). Luckily, this complex procedure is completely transparent to the user.

All the machinery just described does not target the issue of searching a document in the library, which is orthogonal and will be described in next section. However, as soon as we will have a working implementation of a search engine, we will integrate it with the interface simply creating a new HTTP pipeline.

8 Searching and Retrieving

Our aim is to generate an on-line database of machine-understandable mathematical documents, allowing smart searching and retrieving. Since this is also the general aim of the W3C Semantic Web Activity Domain, we decided to re-use the same methodology and tools, mostly based on the exploitation of metadata to improve machine-understandability and inter-operability. Our mathematical documents, though, are already completely formalised and highly structured, so that a large part of metadata can be generated in a completely automatic way. This is important because the main reason for the failure of complex metadata models is usually the lack of suitable tools supporting the actual insertion of metadata: it is often painful to add them by hand, and authoring tools may only provide a surely useful but limited support.

Smart searching and retrieval are not only useful to browse the library, but are also fundamental for the development of proof-assistants to effectively allow re-use of already developed results; even if this seems a trivial requirement for a proof-assistant, nowadays many theorems and definitions are often re-stated by the authors in the new contributions for the mere difficulty to identify the needed notion in the already developed knowledge base.

Typical queries about mathematical objects are the following:

- Search and retrieve all the theories in which a given knowledge item is used/referenced (where a knowledge item is an object as a definition or a theorem).
- Search and retrieve all the theories regarding Linear Algebra.
- Search and retrieve all the theorems that are applicable to a given set of hypotheses.
- Search and retrieve all the theorems whose conclusion matches a given type pattern.
- Search and retrieve all the proofs of a given statement.

Two main kind of basic queries are clearly recognisable. The first one uses informations as names (first query), keywords (second query) or authors which are not specific of the mathematical domain. These queries could be resolved using standard metadata schemas as Dublin Core (`http://dublincore.org`), eventually enriched with domain-specific constraints, for example on the list of keywords.

Although this form of searching is certainly useful for browsing purposes, it is clearly not sufficient during proof-searching, where the names of the needed theorems/definitions are usually unknown. The second kind of basic queries address exactly this problem and allows to search for objects including terms satisfying some given constraints expressed as match patterns on types (second to fourth queries). These kind of queries are extremely expensive and become even more complex if matching on types is defined up to convertibility or some form of isomorphism. Hence, we are going to study what are the metadata that could help this kind of search. The ambitious goal is to automatically generate in a batch process enough metadata to allow the previous queries to be solved on-the-fly.

All the meta-information considered must be associated both to theories/documents and to single mathematical objects. Less expensively, though, some metadata could be associated to whole collections of mathematical objects (e.g. the author of a bunch of related theorems and definitions). Hence, we associate to every theory, collection of objects or single object an XML file (a metadata model) with the relative meta-information. The XLink technology can be exploited to relate the metadata models to the corresponding XML data files.

The metadata model can be conveniently expressed in RDF (Resource Description Framework), a W3C proposal explicitly conceived with this purpose. RDF [5, 6], provides a general architectural model for expressing metadata and a precise syntax for encoding and exchanging these metadata over the Web.

There are two approaches for querying RDF metadata: the more traditional is inserting RDF metadata in a relation or XML database and rely respectively on SQL or XQL[7] for querying; the second one views the Web described by RDF metadata as a knowledge base, applying knowledge representation and reasoning techniques on RDF metadata.

Standard (relational or object) databases are too much rigid to capture the peculiarities of RDF descriptions and schemas. On the other hand, most query languages proposed for semi-structured or XML data, as the XML Query Language, are totally schema-less and cannot exploit the RDF class or property hierarchies and relationships. Moreover querying a RDF data model as an XML instance, needs more than a single XML-QL query due to the fact that RDF allows several XML syntax encodings for the same data model.

Then we need a query language for both RDF descriptions and schemas. A good candidate could be RQL (`http://139.91.183.30:9090/RDF/RQL`) a language for querying Portal catalogs holding multi-purpose descriptions of community resources[8]. RQL adapts the functionality of semi-structured query languages to the peculiarities of RDF but also extends this functionality in order to uniformly query both resource descriptions and related schemas.

While all the other components of HELM described in this paper are already operative and well on their way, the metadata model and its technological infrastructure is still under development.

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 Recommendation, 21 February 2001.
`http://www.w3.org/TR/MathML2/`.

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

[5] Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999.
`http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`

[7] XQL (`http://www.w3.org/XML/Query`) is a query language for semi-structured data.

[8] Community Web Portals provide the means to select, classify and access, in a semantically meaningful and ubiquitous way various information resources (e.g. sites, documents, data) for different target audiences.

[6] Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation 27 March 2000.
`http://www.w3.org/TR/rdf-schema/`

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

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

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

[10] Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., "Content-centric Logical Environments". Short presentation at LICS'2000, June 26-28, 2000, Santa Barbara, California.

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

[12] Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., "Formal Mathematics on the Web". Proceedings of the Eighth International Conference on "Libraries and Associations in the Transient World: New Technologies and New Forms of Cooperation", June 9-17, 2001, Sudak, Autonomous Republic of Crimea, Ukraine.

[13] Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., "XML, Stylesheets and the re-mathematization of Formal Content". Proceedings of "Extreme Markup Languages 2001 Conference", August 12-17, 2001, Montr'eal, Canada.

[14] Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., "HELM and the semantic Math-Web". Proceedings of the 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLS 2001), 3-6 September 2001, Edinburgh, Scotland.

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

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

[17] O.Caprotti, H.Geuvers, M.Oostdijk, "Certified and Portable Mathematical Documents from Formal Contexts", in Proceedings of the First International Workshop on Mathematical Knowledge Management, September 2001.

[18] P. Casarini, L.Padovani, "The Gnome DOM Engine", Accepted Paper at the Extreme Markup Language 2001 Conference, August 2001.

[19] Y.Coscoy, G.Kahn, L.Thery. *Extracting Text from Proofs*. Technical Report RR-2459, INRIA Sophia Antipolis.

[20] Y.Coscoy. "Explication textuelle de preuves pour le Calcul des Constructions Inductives", Phd. Thesis, Université de Nice-Sophia Antipolis, 2000.

[21] Girard, J.Y. "Proof Theory and Logical Complexity". Bibliopolis, Napoli, Italy, 1988. ISBN 88-7088-123-7.

[22] Higham, N.J. "Handbook of writing for the mathematical sciences". Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1993. xii+241 pp. ISBN: 0-89871-314-5.

[23] Howard, W.A. "The formulae-as-types notion of construction", in "To H.B.Curry: Essays on Combinatory Logic", J.P.Hindley and J.R.Seldin editors, Academic Press, London, pp.479-490, 1980.

[24] G. Huet, G. Plotkin (eds). *Logical Frameworks.* Cambridge University Press. 1991.

[25] G. Huet, G. Plotkin (eds). *Logical Environments.* Cambridge University Press. 1993.

[26] J. Kahan, M. Koivunen, E. Prud'Hommeaux, R. R. Swick. "Annotea: An Open RDF Infrastructure for Shared Web Annotations". In Proc. of the WWW10 International Conference, Hong Kong, May 2001.

[27] Knuth, D.E. "The TEXbook". American Mathematical Society, Providence, RI and Addison-Wesley Publ. Co., Reading, MA, 1986, ix+483 pp. ISBN: 0-201-13448-9.

[28] Kohlase, M. "OMDoc: Towards an OpenMath Representation of Mathematical Documents". Technical Report (`http://www.mathweb.org/omdoc/index.html`).

[29] Kohlase, M. "OMDoc: Towards an Internet Standard for the Administration, Distribution and Teaching of mathematical Knowledge". Proceedings of "Artificial Intelligence and Symbolic Computation", Springer LNAI, 2000.

[30] Kohlase, M. "OMDoc: An Infrastructure for OpenMath Content Dictionary Information". Bulletin of the ACM Special Interest Group for Algorithmic Mathematics SIGSAM, 2000.

[31] Martin-Löf, P. "Intuitionistic Type Theory", Bibliopolis, Napoli, 1984

[32] Prawitz, D. "Natural Deduction", Almqvist & Wiksell, Stockholm, 1965.

[33] Schütte, K. "Proof Theory", Springer-Verlag, Berlin, 1977.

[34] Swanson, E. "Mathematics into type: Updated Edition". American Mathematical Society, Providence, R.I., 1999. 102 pp. ISBN: 0-8218-1961-5.

[35] Takeuti, G. "Proof Theory", North Holland Publishing Company, Amsterdam, 1975.