INFORMATION SOCIETY TECHNOLOGIES
(IST)
PROGRAMME

Project IST-2001-33562 MoWGLI

Requirement Analysis
Distributed Digital Libraries

Authors:
For the requirements list: Bernard Schutz, Romeo Anghelache, Christina Weyher
For the appendix: all MoWGLI sites

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

Contents

1 Introduction

Distributed digital libraries are a key aspect of the MoWGLI project. Each of the MoWGLI partners maintains digital stores of information that need to be accessible to other members or to the general scientific public. In this Requirement Analysis we spell out what goals the MoWGLI project has for providing uniform access to a very diverse set of digital records.

To develop these requirements we first distributed a questionnaire regarding digital libraries in MoWGLI. The replies revealed a great variety of activities, with a broad range of depth and scope. These range from proof/theory/mathematical knowledge libraries in Coq and HELM, to the publishing of text that includes mathematics but much more (Saarbrucken and Living Reviews/Golm). The Appendix to this report contains the replies we had, which are the raw material for this analysis. The analysis was then presented to the collaboration at its meeting in July, and the present report follows from that discussion.

The principal requirement is to exploit the common ground among the partners in an activity that is (a) useful to the project's internal goals, and (b) will produce a product for digital libraries could have application in fields not envisioned in MoWGLI. One such application could be to the preparation of documentation for computer programs that numerically solve mathematical problems.

2 Definitions concerning digital libraries

We begin with a set of definitions of terms within the scope of the MoWGLI project.

- A digital library is a collection of digital documents that can be queried for retrieval of content. The library could be distributed over many physical sites. This definition purposely excludes more active interaction with documents, such as proving theorems, creating theories, etc. Such activities can add documents to the library, but a library query itself will not launch an activity to create library content. However, a query may launch a process to combine content extracted from the library into a new output document.

- Metadata are the keys that point to library content and make possible a description of it.

- A query is a request to extract library content, possibly from many libraries. A query can come from any WWW user or site, not just those familiar with how to use the active side of a site offering a library. The query should (1) adhere to a standard mathematical query language, (2) allow a rich set of options for interrogating both text and mathematical content, (3) be "read-only", (4) be able to specify the format of response.

- Retrieval means responding to a query. This should be done automatically, by the library. There should be a variety of user-selected options for the presentation of the retrieved information, as appropriate for the content being retrieved.

- Cataloging enters new content into the library, with the implication that it will immediately be indexed as necessary for responding to queries. This activity will be restricted to trusted people or computer programs, not general WWW users. It may happen automatically as a result of active creation of new content, such as new theorems, new journal articles, etc.

- An index is a table pointing from the metadata values to retrievable content. This index might be explicitly constructed by the library and directly accessed by the query processor, or it could be implicit in a database system if one is used to store the information. Standard databases or new XML-aware databases may be suitable. The index could be dynamical.

- A Schema is a vocabulary of named structures. This vocabulary is used for creating/modifying library items, as well as for checking their consistency and contextual validity at any time during their storage in the library.

3 Requirements

In order to make the MoWGLI project effective, we adopt the following project goals:

- Each MoWGLI site that offers information to WWW users will be able to retrieve that information in response to queries that are posed using a standard language in a standard way, and to refer queries on to other MoWGLI sites automatically. This amounts to providing a mechanism for exchanging metadata between the cooperating libraries.

- MoWGLI will itself specify the query language, since there is no accepted standard language already available that can encompass the project's needs. The language must be flexible and extensible. It must be able to use public dictionaries in order to customize to specific sites. The query language should be an XML language.

- MoWGLI will set minimum standards for how libraries are published. There should be a standard way (API) for computers to contact a library, place a query, and receive a response. This API will interpret queries using the standard query language and mediate the transfer of retrieved data to the user making the query.

- MoWGLI will set minimal standards for how libraries are maintained. The objects of a library are either documents valid relative to a Schema, or objects that can be assembled in such documents. The respective documents/objects are added by the editorial staff/administrators of the library; modifications can be accepted from the users under editorial control; where necessary, the version information should be kept inside the (XML) document itself and any subsequent changes should be structurally appended to it, in full or differentially; the specific way of obsoleting/removing the documents should be flexible/site dependent.

- MoWGLI will set minimal standards for how libraries interact. The manner in which the libraries interact with each other is specified by the metadata set defining the administrative part of the libraries, e.g. locations of a document's components or the way they should be assembled in a meaningful or complete output, copyrights, access permissions, etc. The format of the metadata to be exchanged between libraries should be XML, and conforming to one of the standards currently available: METS [1, mets.xsd], Dublin Core [2, simpledc20020312.xsd], RDF [3, RDF Schema].

  Each of these standards are more appropriate for certain tasks than the others: the format of the metadata gathered from the documents should use the METS , which

was designed for interoperatibility between digital collections . METS is preferable, for library interchange purposes, because its Schema is a better compromise between simplicity and fine granularity of the metadata, allowing also a library to assemble on the fly a document as a result of a peer library query.

RDF is optimal for storing administrative metadata inside the documents or inside objects internal to documents. The Dublin Core set can cover cases where minimal administrative information is necessary. All these standards can be used interchangeably modulo an XSL transformation.

Whatever the choice of the metadata format, a MoWGLI site should be able to assemble a consistent answer to an OAI [4, OAI-PMH] harvester, therefore a specific implementation should follow the specifications of the OAI's protocol for metadata harvesting.

- MoWGLI will define a set of transformations which will provide different representations of the objects queried for, whenever these multiple representations are requested. In the case of XML documents or XML formatted chunks of information, the transformations will be defined in terms of the standard XSL language. The result of such transformations should be browsable online or downloadable for local viewing or printing and should contain links to the library context of the document in question. In other cases, the queries should be able to get a general enough representation of a result, that is, a client independent answer: either a set of HTML pages or a set of pdf files.

- MoWGLI will define best practices in using metadata. The result of a query is a subset of the structured information items set of the library, therefore it should contain: XML links or references, parts of documents or whole documents, possible choices for different representations of these objects.

  Metadata can be stored in the document objects of the library or stored separately. The preferred way is site dependent, e.g. for a library of scientific papers or books, the most effective way of storing administrative metadata is to embed it in the documents themselves, this relieves the library administrators from the burden of a separate process of checking the library's structure consistency.

  When formulating an answer to a query, or whenever an object/document is added or modified, part of the library's metadata can be extracted and assembled dynamically. The same mechanism can be used for creating, one or more, ready to use catalogs, which are available for different categories of subsequent user queries.

- MoWGLI sites will make explicit what is searchable in their libraries by defining such XML schemata and/or metadata vocabularies. These items let the users know which are the formal/canonical parts available to structured queries and which are the informal/free text parts available to nonstructured queries.

  The Schema or vocabulary defining the documents is the tool allowing the validation of their consistency and makes explicit their relationship with a specific library and their peer objects belonging to that library.

  MoWGLI will offer a human-friendly interface for library users to peruse the data/metadata vocabulary which defines the library (e.g. an intuitive, user-friendly graphical display of the XML-Schema/vocabulary/dictionary defining the semantical architecture of the documents in the library).

The objects which form a query's result can be equations, tables, diagrams, images, experimental data, text, or a combination of them, such as a document in its entirety or a table of contents, along with a history of that object wherever it makes sense to provide it. The metadata which describes these items is added at the authoring stage or editorial stage depending on the tools available for this task and the author's willingness to add them and following the structure imposed by the Schema.

- MoWGLI will define a set of representations appropriate for the objects stored in the libraries; when the objects are XML documents, a set of stylesheets will be provided for different devices or for different user interests.
- The implementation of the query/retrieval engine and its user interface will be site-dependent, except that they must conform to the MoWGLI API standard. Thus, the user will not see a MoWGLI-standard interface, because the needs of the MoWGLI sites are too diverse to make a single interface sensible. But behind the interface the process of retrieving information will follow the API. MoWGLI must decide how much commonality of standards for software behind the API is needed among sites, e.g. the format of an index or the use of a standard database.

4 MoWGLI activities for digital libraries

These goals will be implemented by MoWGLI partners as follows.

- The query language will be developed mainly by Bologna and Saarbrücken, coordinated within the Interfaces Work Package (WP). There will be consultation with all MoWGLI partners.
- Metadata standards will be defined within the Metadata WP.
- Query/response API is the responsibility of the Interfaces WP and the Transformation WP.
- Implementation of a query/retrieval engine for searching/authoring text documents will be an activity of Golm and Saarbrucken. In Golm the emphasis will be on LaTeX authoring tools (cataloging) and a mathematical dictionary for the research field of Living Reviews.
- Implementation/dictionaries for other sites will be the responsibility of each MoWGLI site.

References

[1] Metadata Encoding and Transmission Standard, http://www.loc.gov/standards/mets/

[2] Dublin Core, http://dublincore.org/

[3] Resource Description Framework, http://www.w3.org/RDF/

[4] Open Archives Initiative, http://www.openarchives.org/

[5] XML Schema, http://www.w3.org/XML/Schema/

5 Appendix

Answers from MoWGLI groups

A Bologna node answers

A.1 Creation

A.1.1 What does "digital library" mean in the context of the specific applications you are working with?

A digital library is a collection of documents saved in digital format and suitable for automatic processing. In our case, the collection is composed by formal mathematical statements, typically borrowed from available repositories of tools for the mechanization of formal reasoning (proof assistants). The formal collection can be integrated with other data, such as metadata, natural language annotations, examples, images, and so on.

Currently we are mostly making experiments with the repository of the COQ Proof Assistant, that is composed of about 15000 (taking about 260 Mb of disk space after compression) definitions and statements, mostly covering elementary aspects in algebra, arithmetics, and analysis. Each definition or statement is a stand-alone document in HELM. Considering the other data, the HELM library is currently composed of about 65000 documents.

More generally, in the scope of MOWGLI, we could enlarge the acception of digital library to cover not only the documents, but also the information (and the tools) required for its management (e.g. stylesheets).

A.1.2 What is the "content" of your digital library? (A short overview of the data content and format)

The basic unit of information, stored in a XML file, is a formal statement or definition in a suitable logical system. Each logical system has a different DTD, due to the different foundational meaning of its operations and constructions. Other files describe additional informations about these data:

- Metadata files (in RDF), containing meta information of the document itself.
- Annotations, which store presentational human-provided information.
- Views, which are structured collections of (references to) the units of information, suitably assembled for some presentational purpose and possibly intermixed with explanations in natural language and or other media types (images, animations, etc.).

A.1.3 How is content added to library? What are the different pathways how this might happen?

The typical way a document may be (currently) added to the library is by authoring it inside the COQ Proof-assistant, and then exporting the information in XML, using a suitable exportation tool. The same mechanism can be generalized to every other tool for the mechanization of formal reasoning: the only application dependent part of HELM architecture is the exportation module.

A.2 Archiving

A.2.1 What is stored (physically)? Is there information which is generated on the fly on query/retrieval?

All the information is physically stored as XML files. We make a very modest use of data bases for encoding metadata (see subsection A.4.1). Rendering is performed "on the fly" by means of a complex pipe of stylesheet applications. In particular, rendering is roughly splitted in two phases, where the document is first transformed into an intermediate "content" encoding (mostly based on MathML-content) and then transformed into the required final rendering format (currently we support both HTML and MathML-presentation output). Other stylesheets in the pipe are responsible for more specific operations (e.g. hyperlink creations, etc).

Part of our metadata are automatically extracted from documents. This is conceived as a batch process to be periodically performed on the library.

A.2.2 Describe under what circumstances content is being removed/treated as temporary/becomes obsolete.

In a formal setting, removing or modifying a mathematical entity may break the correctness and change the semantics of all other entities referring to it. These operations should be forbidden.

Some documents generated on the fly are temporary. For examples, views of the library generated according to user-preferences.

A.2.3 Do you need/use a versioning system?

No versioning system is currently supported. Nevertheless we could imagine to have different versions of a same document. The delicate point is to understand what kind of support the "versioning system" should provide. In particular, the notion of "conflict" is much more complex (and interesting) than in the case of purely textual documents, due to the semantic description of the information. E.g., creating a new version of a definition should notify the authors of all entities referring to it to give them the possibility to "upgrade" to the new definition.

A.2.4 Are there interrelated entities in the database (i.e. entities whose meaning depend on a context which cannot be automatically reconstructed from the entity alone)?

All entities have a lot of dependencies with each other, but all of them can be traced from the given entity or its associated metadata file.

A.2.5 Is content being stored at multiple physical locations? If not, would you perceive a need for a distributed archive of your data?

Every XML file is stored at multiple physical locations and is identified by an unique URI, which is unrelated to the URL of its physical location. The files are:

- CIC files (in XML). Every object (theorem, definition, axiom and so on) is stored in a different file.

- Metadata files (in RDF). Every object can have one or more metadata files associated to it. The metadata can be on any server, even different from the one on which their data resides.
- Stylesheets. Stylesheets to go from CIC to MathML Content and from MathML Content to HTML or to MathML Presentation are also distributed and identified with a logical name (URI). In this way every user can contribute its own notation and extend the content level, possibly overriding the notation of other users.
- Annotations. Every object can have one or more annotations file, which store presentational human-provided information.
- Mathematical documents as views on the library. For us a mathematical document is just an XML document with references to formal objects that can be expanded on-line by means of stylesheets during the document rendering.

A.2.6 If so, what is the distributed architecture?

Two strong goals of our architecture are:

- Publishing a new file should be as easy as publishing a new web page, without requiring to the user to install any software on the server-side.
- A common browser should be able to consult the library, without requiring any additional software on the client-side.

Thus we are forced to use some kind of middleware. In particular we have developed a component, the *Getter*, with the following features[1]:

- Its interface is the one of an HTTP server. The HTTP GET method is used to invoke the methods of the *Getter*. Thus it can serve the requests of a common browser.
- Its main method, which is `get`, works as follow:
  1. It has one parameter which is the URI of the document that must be retrieved.
  2. A table (actually stored on the disk as a DBM file) is consulted to map the URI to a corresponding URL. If no URL is found, an error is returned.
  3. The document is retrieved using its URL. Any URI can be substituted on demand with a new request to the getter to retrieve the document identified by that URI. In this way relative[2] hyperlinks work as expected.
  4. The modified document is returned to the client and stored in a local cache, that is consulted before downloading the document again.
- The second most important method is `update`, which works as follow:
  1. It consult a local table (a text file) with the list of all the known servers.

[1] The description has been simplified. Nevertheless, it should provide a good idea of the expected functionalities

[2] In the logical space, i.e. the one of URIs

2. In turn it contacts every server in the list and downloads an index file, which is a text file that every server must provide in its root directory and that holds the list of all the URIs of the published files.
3. It associates to every URI in an index file its corresponding URL, computed starting from the URI and the URL of the server it contacted. When a file is provided by more than one server, it chooses one of them accordingly to the preferences of the user (i.e. the relative order of the servers in the list of all the known servers).
4. It fills the DBM file used by the `get` method with the function from URI to URL that it just computed.

Thus the *Getter* provides the functionalities of a proxy combined with the functionalities of a name resolver. Moreover, it could be extended and made more flexible in several ways:

- The list of the known servers could be automatically determined by the *Getter*, that could exchange its list with other *Getter*s it knows of. Such a feature would provide the typical functionalities of a peer-to-peer system.
- The choice of the server contacted to retrieve a document when more than one server provide the same document could be delayed to the execution of the `get` method and then can be made parametric on the network conditions, to achieve load balancing or to minimize download times contacting more than one server at a time.
- The *Getter* could automatically perform an update of the known sites on a regular basis, to avoid getting out-of-synch.
- Versioning mechanisms could be integrated in the *Getter* or could be leaved to an upper abstraction level, provided that URI also hold a version number.

A.2.7 What are the consistency constraints?

An URI must completely identify a document, that must be immutable to avoid having two different copies of the same document on two different servers. This is not a priori in contrast with the possibility of having different versions of a document, but just implies that those different versions must be identified by different URIs. Under this assumption (that can be relaxed by integrating the notion of versioning inside the *Getter*), there is never any consistency problem within replica of the same document stored on different servers.

A.2.8 Is the architecture centralized?

Not at all. The servers, the *Getter*s and the clients can all be on different hosts; documents can be stored on any server.

A.2.9 Are there "trusted" repositories?

There are two different kinds of attacks to the system that a malicious user can attempt. The first one consists of publishing wrong results, claiming that they are true. Formal results can be proof-checked for their validity after downloading them (and all the results they depend on), but this is quite an expensive operation that can not be performed every time a result is referenced, although the *Getter* cache surely helps. Informal results can not be checked at all.

A simple solution is identifying a subset of servers that are considered "trusted". Of course we are left with the problem of authenticating the trusted servers, that can be addressed using standard solutions (e.g. replacing the HTTP protocol with HTTPS).

The second kind of attack consist in publishing on an untrusted server a modified copy of a document, hoping that a *Getter* chooses to download it from the untrusted server.

In some domains (e.g. the certification of security of IT products), attacks of this kind are likely to happen and should be avoided contacting only trusted servers. In other contexts, instead, they may be considered less likely, leading to the possibility to also trust untrusted servers. Finally, when we are dealing with formal objects, untrusted servers can be safely used for performance reasons, provided that the documents provided by them are checked once downloaded.

A.3 Retrieval

A.3.1 Please specify in detail the kinds of queries the API (see subsection 3.4. of preliminary requirement analysis) should be able to deal with for your domain of the application.

Smart searching and retrieval are not only useful to browse the library, but are also fundamental for the development of Proof Assistants to allow effective re-use of previously developed results; even if this seems a trivial requirement for a Proof Assistant, currently many theorems and definitions are re-stated by the authors in new contributions due to the sheer difficulty to identify the needed notion in the already developed knowledge base.

Typical queries about mathematical objects could be as follows:

- search and retrieve all the theories in which a given knowledge item is used or referenced (where a knowledge item is an object such 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 the conclusion of which matches a given type pattern;
- search and retrieve all the proofs of a given statement.

Two main classes of basic queries are clearly recognisable. The first one uses general information such as names (first query), keywords (second query) or authors, that are not specific to the mathematical domain.

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 typically unknown. The second kind of basic queries address exactly this problem and allows the searching of objects including terms satisfying some given mathematical constraints (second and third queries).

Our API should give access to a searching engine capable of retrieving resources[3] on the basis of mathematically relevant information. Identifying this information is an interesting problem. A starting point is surely a detailed description of the dependencies between resources (such as what notions a statement is about and where these notions are used in the

[3]By *resource* we mean a mathematical notion stored in an entire CIC file or in a part of it.

statement, i.e. for instance in the premises or in the conclusion) because mathematics can be naturally structured as a set of items (definition, theorems or whatever) linked by logical dependencies and this seems to be one of its peculiar aspects.

The dependencies consist of direct and inverse references, and their description should at least include an indication of the position where they occur in the referrant resource. These positions must have a clear mathematical meaning and currently we are using the following classification: "in a premise", "in a conclusion", "in a proof"; a distinction is also made if the reference occurs in the main position of a premise/conclusion, or does not.

Of course it may be the case that a sharper classification is necessary, or that we need a deeper degree of accuracy in describing the dependencies.

In any case the basic queries are of the kind: retrieve the resources whose dependencies satisfy a given set of constraints (typically a constraint for a dependence is that it must be a direct or inverse reference to given resources, appearing in specific positions).

These queries are expressed in a query language, which we call MathQL, with the following design goals:

- Its syntax and semantics must abstract from the concrete representation format of the queried data, i.e. XML in our case.
- Its core features must be designed on the assumption that we are treating mathematical information. In particular it must be possible to express the constraints mentioned above.

Retrieving resources on the basis of general information (i.e. not specific of the mathematical domain) is also possible and this includes, but is not limited to, standard Dublin Core metadata. For instance we can retrieve all resources whose URI[4] matches a given pattern[5].

Obviously MathQL supports these queries and also supports standard operations on query results such as union, intersection and sorting.

A.3.2 Please specify what's going to be retrieved.

The result of a query is a list of resources specified by their URIs so what a query produces is actually a dynamic view of the library (see A.1.2) containing the retrieved items.

A.3.3 How should the results be presented (HTML page, math code ...)?

As the results of a query form a view of the library, they are presented as such, using our "rendering on-the-fly" engine (see A.2.1) which produces HTML or MathML Presentation.

A.4 Implementation

A.4.1 How is search metadata being stored in the library and associated with the documents?

Our aim is to generate an on-line data base 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

[4]The URI of a resource can include a fragment identifier.

[5]This query is significant provided that the URIs have a meaningful structure.

based on the exploitation of metadata to improve machine-understandability and interoperability.

The standard format for metadata proposed by W3C is RDF (Resource Description Framework). RDF provides a common model for describing the semantical information on the Web corresponding to different domains of application: RDF allows anyone to express assertions (i.e. statements as subject-verb-object records) about everything (i.e. resources) on the Web. In particular, RDF provides general structural constraints for expressing unambiguously metadata in terms of properties of resources and a precise syntax for the encoding and interchange of these metadata over the Web. RDF schemas declare properties with their possible contraints, and define relationships between properties and resources.

RDF, as HELM does, heavily relies on URIs: URIs represent the way both to identify resources and to associate metadata with data.
Moreover, RDF follows the same consistency constraints of HELM (see subsection A.2.7), encouraging the reuse and extension of metadata semantics among disparate information communities.
Least but not last, one of the main design goal of RDF, which has also influenced its exploitation in HELM, is scalability: the RDF three-part records fit very well for huge set of (meta)data, making easy to handle and process them.

Since our mathematical documents are already completely formalised and highly structured, a large part of metadata can be generated in a completely automatic manner. Currently, HELM automatically extracts the metadata about forward (i.e. references from the document) and backward (i.e. references to the document) dependencies between entities (see subsection A.3.1), exploiting RDF (XML syntax) as storing format.
Also, the general Dublin Core meta-information is manually inserted and encoded into RDF. HELM provides several RDF Schemas to define both a specific vocabulary for HELM mathematical resources, and a vocabulary for metadata about general resources, say the Dublin Core elements.

All the considered meta-information is associated both to theories/views and to single mathematical entities. Less expensively, though, some metadata is associated to whole collections of mathematical objects (e.g. the author of a bunch of related theorems and definitions). URIs are exploited to associate a RDF file (a metadata model) to the corresponding XML data file or directory (e.g. view, collection of objects, single entity).

HELM has about 31000 metadata files for Dublin Core, backward and forward information, taking respectively about 172Kb, 100 Mb and 81Mb of disk space. The automatic generation is performed as a batch process, which requires about 5 hours running on a Pentium III 600 MHz.
To access metadata we rely on the PostgreSQL DBMS. Inserting metadata files in the data base requires about 2 hours, running on a Pentium III 1 GHz and generating a data base of about 416700 statements.

B Nijmegen answers

B.1 Creation

B.1.1 What does "digital library" mean in the context of the specific applications you are working with?

For us a digital library is a collection of Coq files (.v files) containing mathematical theories developed in Coq and meta-information allowing documentation (in LaTeX-format) to be automatically generated from them. So to be more precise, the .v files contain the usual Coq code (definitions, lemmas and tactic scripts) plus meta-information, 'hidden' in Coq-comments, which allows a special script to generate LaTeX-documentation.

B.1.2 What is the "content" of your digital library?

Presently our digital library consists of a directory structure that contains:

- a collection of Coq (.v) files;
- some tools (scripts) from which other files of different formats can be generated.

The files generated are:

- compiled Coq files (.vo), which can be imported (as already developed mathematical theories) for use in new work;
- LaTeX files which document most of the development

B.1.3 How is content added to the library? What are the different pathways how this might happen?

At present, only authorized users (i.e. who have access to our CVS repositorty) can add content to the library. This is done through the standard CVS procedure: each user develops his own work (.v files) using the standard tools (typically Coq running under X-Emacs with the Proof General interface), and including the necessary comments so that the documentation files can later be automatically obtained. Then he uploads his files to the CVS repository.

B.1.4 Regarding scalability: How many entities does your digital library contain (order of magnitude)? What part of it is or is expected to be equipped with markup?

Our digital library contains around one hundred .v files, divided in around ten directories corresponding to the mathematical domains that they cover. Besides these there are ten script files and other support tools (including a .ml file to compile our own version of the Coq proof assistant with extended decision procedures, which is needed for our work) which, when invoked, automatically generate some other one hundred files.

Some of the directories in the library are labelled as "work in progress". The files in these directories are typically being worked on, in contrast with the bulk of the library which is mostly stable. Work-in-progress files are usually not yet commented. All stable files include comments that generate documentation, and are suitable for other kinds of markup.

B.2 Archiving

B.2.1 What is stored (physically)? Is there information which is generated on the fly on query/retrieval?

The main repository contains *only* the .v files containing annotated Coq code. (So a CVS update does not update the .vo files.) All other files are generated from these.

Presently, due to the small number of users, each person has a copy of both the .v files and all generated files on his computer, as the compiled (.vo) files are needed for further theory development and the .tex files provide the only documentation available. However, we envisage a system where the documentation is only temporarily generated and nowhere stored. This does not seem to be feasible for the .vo files, as they take some time to produce (at the moment, compiling the whole library takes around 45 minutes on a 900MHz computer with 384Mb RAM) and are constantly needed for work.

B.2.2 Describe under what circumstances content is being removed/treated as temporary/becomes obsolete.

As we have previously said, the library is divided in two kinds of directories: the developed (stable) part and the work in progress. Files in directories of the second kind are unstable, being frequently changed (every day), merged together, split or otherwise changed as information in them is moved around and extended.

There are two kinds of circumstances which lead to a significant change of content in the main (stable) part of the library. Typically, at some stage one of the work-in-progress directories becomes stable (that is, when it is considered "finished" as a chapter of mathematics), and at that point it is incorporated in the main library. This process often also implies some moving around of information, as some results (e.g. lemmas that apply more generally then just to the specific domain) may make more sense in other directories because of the subject they refer to.

The other kind of circumstances are mathematical, when a lemma or a set of lemmas is replaced by something more general. This will usually not only imply removing the original lemma(s) and adding the new one(s), but also changing proofs which depended on those lemmas—and will thus amount to a significant change in the library as a whole.

B.2.3 Do you use/need a versioning system?

Presently we use CVS, as it allows us to keep track of changes. Some kind of versioning system seems to be needed, as big changes are periodically made in the library as a whole for the reasons above described.

B.2.4 Are there interrelated entities in the database?

The whole database is deeply interrelated. The Coq dependence mechanism requires that compilation be done in a standard order, because files inside each directory are built on top of each other.

From a mathematical point of view, the division into files is somewhat arbitrary, as it may be done for mathematically meaningless reasons, such as the files becoming too big or taking too long to compile. So we prefer to think of our main entities as being the directories—which correspond to distinct mathematical subjects or chapters. Also from this point of view

there are obvious relations and dependencies between entities, as mathematical subject are interdependent. These relations seem to be mathematical and are therefore unavoidable.

B.2.5 Is content being stored at multiple physical locations?

Content is stored in the main CVS repository, although users typically have their own personal copy of every file. However, we envisage a future where development is done by different people in different locations but inside a common project, and in this situation it would make more sense to have a distributed architecture where there is a central server with *information* on where each directory is stored (presumably this will be at the place where development on that specific branch of mathematics is being done).

B.2.6 If so, what is the distributed architecture?

See previous answer.

B.2.7 What are the consistency constraints?

Our consistency constraints are basically the Coq requirements (all the files have to be compiled with the same version of Coq) and CVS requirements (if two people simultaneously change the same file the second one will not be able to add his changes to the library).

In case of a distributed architecture, another 'consistency' constraint arises, which is the use of names: if two developments use the same name for a specific notion, a conflict arises if one wants to use these two developments simultaneously. Now the CVS repository serves as the central library, and this singles out these naming conflicts at an early stage.

B.2.8 Is the architecture centralized?

Yes, right now it is, by using CVS.

B.2.9 Are there "trusted" repositories?

We feel that at the present moment this is not a relevant issue.

B.3 Retrieval

B.3.1 Please specify in detail the kinds of queries the API (see subsection 3.4. of prelimary requirement analysis) should be able to deal with for your domain of the application.

See previous report.

B.3.2 Please specify what's going to be retrieved.

Annotated Coq-code, i.e. (parts of a) .v file with meta-information (basically LaTeX-code) for explanatory purposes.

B.3.3 How should the results be presented?

This will strongly depend on the use one wants to make of the information: (i) if the purpose is to help in an interactive proof development, Coq code should be returned (one has to know the names and the precise forms of the statements), (ii) if the purpose is to help writing documentation or a paper, LaTeX code should be returned, (iii) if the aim is to present the work on the internet, then some form of HTML code should be returned (generated from the .v file).

B.4 Implementation

B.4.1 How is search metadata being stored in the library and associated with the documents?

As comments in the Coq code.

C Saarbrücken node answers I - MBase group

C.1 Creation

C.1.1 What does "digital library" mean in the context of the specific applications you are working with?

It means a repository with the capabilities of retrieving, storing and structuring specified forms of content. For our context of proof planning and automated theorem proving this means:

- machine readable input and output
- human readable content possible — but with recognizability for automated services
- connectivity via standard protocols
- retrieving possibilities for the content

The answers to question have to be regarded with respect to our development of MBase — a structured repository of mathematical content.

C.1.2 What is the "content" of your digital library? (A short overview of the data content and format)

The documents in MBase have to be in `OMDoc` format.
The typical are mathematical texts — formal and informal. Especially theorems and (parts of) theories as such may appear as results from automated services.

C.1.3 How is content added to library? What are the different pathways how this might happen?

Content is added through input of `OMDoc` documents.
Such documents can be loaded by two possibilities:

1. direct loading of a file by a given URL,
2. connection through XML-RPC.

C.1.4 Regarding scalability: How many entities does your digital library contain (order of magnitude)? What part of it is or is expected to be equipped with markup?

At the moment we have more than 2000 thousand `OMDoc` files. The axioms, hypotheses or definitions are about 4000 (more than 1000 hypotheses or theorems). It is planned to enlarge this in the near future to more than 20 thousands. For mid term future we plan to add content of Mizar.
MBase was realized by an efficient relational database for scaling abilities which are hardly surpassed by todays portable software. Some features of MBase rely on the used programming language Mozart, because of features missing in SQL. As long as one stays with a single system this does not matter because the internal Mozart mechanisms use less time than the database access. Scalability in the sense of multiprocessing or concurrency is not implemented for MBase now, but can be done within Mozart.

A technical need for such methods can be seen by an amount of data which surmount the HELM data by more than ten times.

C.2 Archiving

C.2.1 What is stored (physically)? Is there information which is generated on the fly on query/retrieval?

The information is physically in a relational database. The tree–like structure of the underlying XML files is reconstructible.
The output of search operations (included pattern matching) on the connected database can be redirected to a HTML–Browser which acts as a user–agent. Adding files (manually) takes immediate effect which (of course) can change output for queries used before — the underlying database gets changed.

C.2.2 Describe under what circumstances content is being removed/treated as temporary/becomes obsolete.

At the time in MBase there is no removing of content possible. One has to rebuild without the content which should be removed.
Such features are desirable, but such matters are not trivial for structured content. One has also to be aware that in the long term rights of authors have to be handled in some way (or if not, that a manual treatment would be necesssary).

C.2.3 Do you need/use a versioning system?

At the time there is no versioning in MBase.
But there is a need for versioning of stored (and changed) `OMDoc` content.
The implementation of Development Graph features only partially solves this problem.

C.2.4 Are there interrelated entities in the database (i.e. entities whose meaning depend on a context which cannot be automatically reconstructed from the entity alone)?

Structured content implies the existence of interrelations. Since symbols are basic entities such could also be stored without interrelations. Formal content is supposed to be based on previously defined symbols. At least at this point interrelations approach.
Though one could also store simple text such that there would be no interrelated items.

There are 2 modi now:

1. a sloppy modus, there undefined entities like symbols need not be defined at loading (a proxy item is generated),
2. and the other strict (default) modus demands the resolvability of incorporated notions and thus the interrelations.

The Development Graph concentrates only on the interrelations between theories what is too rough for the intended question. But MBase has not the demand of a tree–like structure of the theories like Mizar.

C.2.5 Is content being stored at multiple physical locations? If not, would you perceive a need for a distributed archive of your data?

MBase stores its content within one location. One can start many MBases.
We believe into data exchange between different MBases. There should also be a possibility to create a small special purpose MBase by retrieving content from large grown ones.
But features supporting distributed MBases have to be developed (an aim within MoWGLI).

C.2.6 If so, what is the distributed architecture?

In a strict sense there is no distributed MBase, but content may be distributed between MBases.

C.2.7 What are the consistency constraints?

In `OMDoc` id attributes have to be unique as long as it concern top items. MBase keeps an eye on this blocking reloading because of superflousness.

C.2.8 Is the architecture centralized?

A single MBase is located on a central place. Our idea on this is more like the world wide web: many MBases communicate with each other (via URIs) and bild a network of many single but central services.

C.2.9 Are there "trusted" repositories?

No. There is no intention on this. If one wants this an MBase has to be put behind a firewall or other network separation techniques (or within an intranet).

C.3 Retrieval

C.3.1 Please specify what's going to be retrieved.

A query results in a list of filtered content. The single parts of such results consist of records for items asked for. That means that the features are associated with content which belongs to an answer of a query.

C.3.2 How should the results be presented (HTML page, math code ...)?

Common ways of presentation are preferred:

1. (pure) XML
2. MathML
3. OpenMath
4. HTML
5. LaTeX

C.4 Implementation

C.4.1 How is search metadata being stored in the library and associated with the documents?

In `OMDoc` metadata are a (main) subtree of the document (or part of it) in question. Hence MBase stores metadata always in connection to a given item. Of course there are metadata specific entries in the underlying database.
At now there are only common search possibilities for metadata. MBase shall get support for special for combined search in metadata and text content.

D Saarbrücken node answers II - ActiveMath group

D.1 Creation

D.1.1 What does "digital library" mean in the context of the specific applications you are working with?

The ActiveMath learning-environment is a web-server meant fir dynamically presenting dynamically semantically encoded mathematical documents for learning purposes. ActiveMath provides such facilities as personalized presentation, user-model, and a content choice according to the knowledge of the user.

ActiveMath can be considered as a digital library in the usage of this software as a middleware to present mathematical content served from a Web of repositories assisting the user in the process of reading (as it infers the user's knowledge from the reading), and it offers the personalization of content presentation.

D.1.2 What is the "content" of your digital library? (A short overview of the data content and format)

The content is a set of mathematical items, each having unique identifiers. Items are elementary building-blocks such as examples, definition, proof, or theorem. Their encoding is, currently, expected to be in `OMDoc` s which provides the necessary level of semantics for far-reaching content re-usability, for the personalized presentation, and for the necessary structure for extensible metadata annotations. We have worked on a set of extended metadata annotations for the pedagogical applications of ActiveMath.

Presumably, the flexible presentation engines of ActiveMath could be ported to different content encoding. In particular, we are investigating the possibility to have presentation-MathML or even TEX-encoded content. This will clearly have consequences and restrictions on the presentation capabilities (e.g. only a TEX-aware presentation-engine could present a TEX-encoded content).

The *item* level, however, cannot be avoided within ActiveMath. The items are meant to be relatively independent from one another. This brings the ability, for example, to pick content from an existing collection to achieve a particular goal which might be only a side-goal of the collection.

For a content collection to be served, two more ingredients are needed: presentation information for each of the symbols defined in this collection and *table-of-contents* which provide views on the collection in the form of list of items to be presented.

D.1.3 How is content added to library? What are the different pathways how this might happen?

The current database-engine is a RAM-based fallback solution which considers its content as a set of directories from which to load. These directories, can, typically, be under control of a CVS system.

When using a real database, such as MBase, the ways we imagine to add some content will be using a tool that validates the content before it is added. Removals and modification of content will use similar tools, they should be aware of rights to perform modifications and warn the user about possible inconsistencies generated and about items that are affected by the changes.

D.1.4 Regarding scalability: How many entities does your digital library contain (order of magnitude)? What part of it is or is expected to be equipped with markup?

It is expected that ActiveMath accesses several databases at the same time, with items that can refer to each others. Each might want to contain such thing as 100'000 items. Currently, however, the solution allows only a small content (about 2'000 items).

The macro-structure, going until the item level and their relationships, should be completely equipeed with mark-up, their aim is to be used by tools to provide an enhanced presentation and tools to plan content. The micro-structure, aside of references, might contain non-marked-up content.

D.2 Archiving

D.2.1 What is stored (physically)? Is there information which is generated on the fly on query/retrieval?

As mentionned, the current document format is the `OMDoc` format. It is the way we expect content to be stored as it allows a great flexibility in the packaging and keeps content at its most general stage,moreover, `OMDoc` collections can easily be reused, as their only *context* is the set of their references (this is to be compared with the much more process-oriented context in use, for example, in the TEX formats).

On retrieval of content, the user is expected to be able to read the documents. For this task, a presentation process is applied which converts the content to a readable format (currently HTML and PDF, soon xHTML+MathML and SVG). This process does not add or generate information, except annotations attesting the user model values representing the user's knowledge, it is a (lossy))transformation process for which an extensible architecture with caching is being built for ActiveMath.

D.2.2 Describe under what circumstances content is being removed/treated as temporary/becomes obsolete.

The decisions to change content is left to the authors and editors of content-collections.

D.2.3 Do you need/use a versioning system?

Versioning will be needed soon. Authors and editors should be helped by tools to support the user manipulating the collections. Among their functions, the abilitiy to inspect relationships so as to decide on appropriate modifications is expected.

D.2.4 Are there interrelated entities in the database (i.e. entities whose meaning depend on a context which cannot be automatically reconstructed from the entity alone)?

For an item's textual content to be presented, the only information needed is the item itself, and, for each of the symbol occurring, the presentation and the title. Hence, for the purpose of presentation the answer to this question is yes.
For an item to be part of a course generation (which creates a *book* with the content needed for the user to master a given set of goal-items), the items it is related to need to be extracted

and, for a definition, its symbol will be extracted, for a theorem, its proofs is extracted. In the sole process of course-generation, the metadata information would be enough.

D.2.5 Is content being stored at multiple physical locations? If not, would you perceive a need for a distributed archive of your data?

Content is currently stored in one single database. The distribution is justified at least by the fact that content should remain administrated by the responsible authors/editors. This is particularly important for the appropriate evolution of content where enrichments and corrections are essential to be centrally administrated (centrally for the content collection).

D.2.6 If so, what is the distributed architecture?

The flexible database connection architecture of ActiveMath allows several databases to be connected. Moreover, the user-oriented features of ActiveMath make it a prime choice as a *middleware* system serving to a browser a Web of databases of mathematical content-collections.

In a distributed environment, the versioning system adds another degree of difficulty. It is probable that multiple versions will have to be served by any database and, as such, each requests should include it. Solutions to milden the need to serve (quickly) all versions is probably to be found, as otherwise the content storage would be quickly overloaded (compare, for example, with the speed of CVS servers), a subscription mechanism is an example of such.

Apart from distribution of content repositories, The ActiveMath architecture can be distributed, allowing, for example, the presentation and course-generation engine to live on multiple servers whereas the user-model would live on only one server.

D.2.7 What are the consistency constraints?

As ActiveMath handles mostly textual content, consistency in the mathematical sense cannot be ensured. Reference consistency (that is the verification for existence of reference-targets and their appropriate types), just as in the normal hypertextual Web, is the only part that can be checked.

Note, however, that `OMDoc` was designed to encode items in both a formal and a textual way. If the parallel between the formal and textual encoding of each item can be ensured (it can only be done manually), the proof checking of a content can also be achieved.

D.2.8 Is the architecture centralized?

The ActiveMath server centralizes, for each user, the caching, serving, transformation, and course-generation capabilities. The user-model is accessed remotely. Its middleware role fits well as a server for a relatively small group of people (for example a classroom or a team of researchers). The content repositories themselves are expected to be distributed with management happening on the content-servers.

D.2.9 Are there "trusted" repositories?

Currently not. If a versionning system is allowed, the simple trust of finding the same content for the same query is the only one we expect to happen. The notion of subscription, if

applied, might also develop the notion of trusted repository, where a subscription, if honoured, would guarantee a client to provide an appropriate content. Documenting the changes within a version evolution (possibly in a formal way), might be another approach to ensure the trust towards new version of the content; for example, a change-annotation stating that only additions or typos-corrections have been performed could, within a given subscription, allow the pointers to upgrade their version-numbers automatically.

D.3 Retrieval

D.3.1 Please specify in detail the kinds of queries the API (see subsection 3.4. of prelimary requirement analysis) should be able to deal with for your domain of the application.

In terms of API, the ActiveMath server queries content from the database using a dedicated interface, mostly tunneled through an XML-RPC interface. This interface is used for reading only and can be basically split in two types: the metadata queries and the content queries. The latter are simple extraction queries, requesting the textual, formal, or metadata part of the item, given by an ID. The former are factual queries returning the relationships definition.

On the user side, the ActiveMath environment offers the following methods to retrieve content:

- the *dictionary* allows a user to search for any substring through the text, it returns a list of found items names which are then displayed. The dictionary is also the way ActiveMath uses to display single items: among others a presentation of symbol declarations can be viewed when clicking on the symbol in a formula presentation. This front-end is rudimentary and we expect to be able to use the much richer queries that Mowgli deliverables will provide. Presumably, user-interfaces for such languages will have to be built.

- *recorded-books* can be read: they are specified by table-of-contents, created by an author, editor, or instructor. The content is then presented as pages of a book which can be browsed.

- *course-generation* can also be applied: using a list of possible concepts to be presented (another kind of table-of-contents), the user prompted to choose goal concepts to be learned about. The course-generator then plans the resulting book using the metadata information of each items and using *pedagogical rules* to control the style of the generaed book.

- while reading a book, the user may also take advantage of the currently prototypical suggestion mechanism, suggesting content to be re-read or exercises to be performed.

D.3.2 Please specify what's going to be retrieved.

See previous question.

D.3.3 How should the results be presented (HTML page, math code ...)?

ActiveMath presents content in a user-readable way. Currently, the HTML format using Unicode-encoded characters for mathematical symbols is used. For printing purposes, a page

can also be printed in the PDF format. More output formats are being worked on, including the MathML-presentation format, and a SCALABLE-VECTOR-GRAPHICS output of a LaTeX presentation.

It is important to note that these presentations are generated on the fly enriched with information that make the reading and navigation interactive. Among others, in the HTML presentation, the name of a symbol is displayed in the status-bar of the browser, links are created to allow the opening in the dictionary, and, recently, a feature has been added to allow the copy-and-paste between formulas presented in the content to the interactive exercise consoles connected to a mathematical system for which OpenMath phrasebooks exist.

D.4 Implementation

D.4.1 How is search metadata being stored in the library and associated with the documents?

Currently the database serving content is the same database that serves the associated metadata. This is a problem for speed as metadata queries (the relationship queries) are heavily used for course-generation (we recently experienced no less than 1500 queries for a course generation, which could be achieved on our current fallback within about 3 minutes). Moreover, the current query mechanism is elementary.

It is our intent to take advantage of the developments of the semantic-Web research, where knowledge-bases can be queried using highly sophisticated queries. For example, several of them support queries that can go arbitrarily deep through a sequence of relations. We also expect to be able to query such information the amount of items that are (transitively) related to one another. Presumably, our course generation and suggestion engines will then support much more expressive pedagogical rules.

E MPI node answers

E.1 Creation

E.1.1 What does "digital library" mean in the context of the specific applications you are working with?

For our side, the digital library will contain information on mathematical objects which are extracted from articles published in *Living Reviews in Relativity*. These articles provide reviews on different areas of research in relativity. Mathematical objects included there are equations, tables, images, diagrams. The content will be focused on mathematics typically associated with general relativity, astrophysics and related areas (i.e. differential geometry, tensor calculus, partial differential equations, physical content like Riemann tensor, Planck law...).

E.1.2 What is the "content" of your digital library? (A short overview of the data content and format)

We are currently in the process of deciding on the format and the encoding of the mathematical objects, as appropriate markup has not been applied yet to our documents. Entities of the digital library will presumably be encoded in XML and/or MathML.

Potentially the database will involve storing individual pieces of the articles (equations, theorems) as individual entities. The process of rendering an article of performing a search would then involve piecing together this entities in an appropriate way.

The articles can also be kept in a single physical piece without loss of speed. XML native databases (Tamino from Hamburg, commercial; or Apache xindice, free software) can query a set of XML files for specific types of elements or attributes through XPath. XSLT can rearrange the structure of the XML article for rendering purposes, on the fly. This way, the burden of assembling and dissasembling articles, and of handling XML files through the traditional database approach, can be avoided.

E.1.3 How is content added to library? What are the different pathways how this might happen?

The content will be extracted from *Living Reviews* articles. Authors of the articles should be enabled to provide the appropriate metadata information, that is neccessary to extract meaningful XML statements. Thus we will need to develop a LaTeX-based authoring tool, as well as a standard dictionary for the mathematical objects encoded.

That is, adding a set of macros to LaTeX, enabling authors to semantically structure the document, and a launch option or a lexer to extract the XML content from the TeX source. The dictionary used should be a an XML-Schema close to OMDoc.

Content will be added to the repository during the editorial processing of articles. A workflow model that integrates this task into the current article processing with a minimum of editorial effort will need to be developed.

E.1.4 Regarding scalability: How many entities does your digital library contain (order of magnitude)? What part of it is or is expected to be equipped with markup?

Currently 31 review articles are published in *Living Reviews in Relativity*, with another 50 in preparation. Articles can vary greatly in the type and amount of semantic markup that they require. Compare for example Alan Rendall's review on "Local and Global Existence Theorems for the Einstein Equations"(`http://www.livingreviews.org/Articles/Volume5/2002-6rendall/`),which is a descriptive overview, with the detailed mathematical treatment of Luc Blanchet's "Gravitational Radiation from Post-Newtonian Sources and Inspiralling Compact Binaries"(`http://www.livingreviews.org/Articles/Volume5/2003-3blanchet/`). Overall, the size of the library is not expected to be large, since we are only publishing at a of 6 to 10 articles per year for which markup will have to be created.

E.2 Archiving

E.2.1 What is stored (physically)? Is there information which is generated on the fly on query/retrieval?

All information on the mathematical contents of the articles should be stored physically as static entities. Certain types of presentation will be generated on the fly on retrieving an article or performing a search. Conversion from marked up mathematics to alternate output formats such as TEXor computer algebra system language would be done on request.

E.2.2 Describe under what circumstances content is being removed/treated as temporary/becomes obsolete.

Due to the editorial concept (`http://www.livingreviews.org/Info/AboutLR/concept.html`) of *Living Reviews*, authors update their articles regularly. Each article update is a major revision of the previous version, including e.g. the extension of existing sections, covering new aspects, updating reference lists, or adding further visual material. An article update is treated as a new publication, which undergoes the full refereeing process and is identified by a unique publication number. The updated article remains at the same URL on the server and is still available when the new version is being published. But the old article version is not longer linked from the journal's article index pages, instead users are directed to the new article version. Consequently, we would also need identifiers for the mathematical entities in the repository, which allow users to distigush between content from updated articles and content from articles that are current. We will need to define, whether associations between "the same" entity of different versions may provide useful informtion (e.g. this equation has underwent certain changes during article evolution?).

Living Reviews has also introduced an amendment scheme, which allows our authors to do *minor* corrections and changes on the current article. Unlike errata in printed journals, amendments are directly included in the respective article location, together with a link to the old version of this corrected paragraph, equation, etc., and a description of the respective changes. For an example see Greg Cook's article on "Initial Data for Numerical Relativity" (`http://www.livingreviews.org/Articles/Volume3/2000-5cook/`). In the end, amendments may result in equations or other mathematical content in the database that can be considered as obsolete, since it has been corrected (and a new version is being added to the

repository). Identifiers should also be able to carry this kind of information.

We can provide, at the end of the query, a link to the older version. There is no need to generate and store IDs for the articles/entities, we can use XPointer / Xlink to locate deep inside the structure of outdated documents.

(For a concise overview on 'living' article evolution, the amendment scheme and the appropriate documentation please refer to (`http://www.livingreviews.org/Info/Help/viewarticle.html`).

E.3 Do you need/use a versioning system?

Living Reviews uses the Concurrent Versioning System (CVS) for the processing and publication of its articles, and maintaining a revision history.

E.4 Are there interrelated entities in the database (i.e.entities whose meaning depend on a context which cannot be automatically reconstructed from the entity alone)?

Presumably interelationships will arise as a result of search queries for individual concepts or material which has been logically or typographically grouped as in the retrieval of an article. These interrelationships could on the one hand be very simple by referring to keywords or metadata attached to individual database entries. On the other hand, more sophisticated retrievals can be imagined in which for instance common subexpressions in an equation are searched for. This last application may be ambitious for the current project, but we would like to think ahead and at least lay the groundwork for future considerations.

Individual entities should be complete on their own either as individual equations or standalone documents.

E.4.1 Is content being stored at multiple physical locations? If not, would you perceive a need for a distributed archive of your data?

Content of the digital library may be stored at different physical locations, since the *Living Reviews* web server is being mirrored on several servers worldwide. However, there is always only one central server which provides the reference version for the current database status (and only there is content being added). Thus the library has characteristics of both a centralized and (one-way) distributed architecture.

E.4.2 If so, what is the distributed architecture?

We run a central web site from which up to 40 mirrors take content. Mirrors are operated by the European Mathematical Information Service (EMIS).

E.4.3 What are the consistency constraints?

The frequency of mirror updates is under control of the individual site. Content of the site tends to change in large blocks (e.g. addition of an article) every several weeks.

E.4.4 Is the architecture centralized?

New content is only added directly to the main site, from which mirrors update on their own time scale.

E.4.5 Are there "trusted" repositories?

As an electronic journal, *Living Reviews in Relativity* already meets high standards regarding the quality and reliability of its published content. In particular, both are guaranteed through the following set of quality management and security mechanisms:

- *Living Reviews* articles are invited pieces, written from experts in their fields.
- All articles are peer-refereed
- All articles are proofread.
- During publication, *Living Reviews* uses a versioning system which ensures consistent tracking of changes the document source underwent and provides always a reference version (evenif somebody hacked on our server changing output files)
- All files associated with the journal (files on webserver, document sources in processing environment) are regularly backupped according to institutional policies

Thus we believe that a digital repository emerging from the mathematical contents of *Living Reviews* articles can be considered as a trusted repository. All published content is being controlled during the editorial processes of the electronic journal, and only content that underwent the complete editorial process is being added to the repository.

E.5 Retrieval

E.5.1 Please specify in detail the kinds of queries the API (see section 3.4. of prelimary requirement analysis) should be able to deal with for your domain of the application.

Users should be able to search through the library with a specially designed search engine. Several "first level" search categories will need to be defined, which give information on the "nature" of the mathematical contents that can be searched (e.g. equation, concept, definition, image ...). Each of these categories allows specific features that can be searched for, e.g. one can search for captions in category "images". The objects in the library will be indexed in an hierarchical way, allowing each object to belong to several categories.

The administrative information can be stored also inside the article, best through RDF. This way, the administrative information and the rest of the contents can be accessed/selected uniformly through XPath queries and XPointer / Xlink, using the same XML database/indexing software.

The API should be able to deal with adiacency functions, i.e. enabling the user to search for entities that appear in a given order in a database entry. This type of search could also be applied across entries, as in the case of an article which might be decomposed in the database into a number of ordered entries.

It should be possible to have certain associations between the entities.

The details of the mathematical searching interface are still to be considered. Applications that one would definitely like to see would be searching on equations by their name, or for individual defined mathematical quantities (e.g. the Riemann tensor) within the full set of equations in the library.

E.5.2 Please specify what's going to be retrieved.

Retrieval can take a variety of forms, from a full article to individual sentences, paragraphs or equations. Each of the latter would include a link back to an originating article which is likely the end goal of any user search of an electronic journal.

The layer of structure queried for is customizable directly from the search interface, using the XPath, the administrative information (stored as an RDF header inside the article) is such a layer.

E.5.3 How should the results be presented (HTML page, math code ...)?

The initial response from the search engine will be HTML, with links to other types of data, for instance TEX source for an equation or, optimistically, computer algebra input, or MathML. Since the primary display format of our articles is HTML, there will also be a link to the originating article.

E.6 Implementation

E.6.1 How is search metadata being stored in the library and associated with the documents?

We are currently in a very early design stage of the library, and many of these issues are yet to be decided upon. However, our early perspective can be defined in the following paragraphs.

Metadata is composed of administrative information and content annotations. The administrative information is added in the document by the editorial staff. The content annotations are created by the authors of the documents and, whenever necessary, by the editorial staff with the authors' agreement. If the documents stored in the library are XML documents, then the metadata is formed by the instances of the XML vocabulary or Schema used to define and create that document .

Formal queries, as well as informal queries, should get a meaningful result. Therefore, the formal part of the annotation metadata should have a well definedstructure according to a vocabulary, on which structured queries can be applied, while the rest, that is, the informal part of the annotation metadata, can be queried just in terms of string inclusion/comparison.

Most probably, the metadata will be stored inside the articles themselves as RDF; RDF is XML, so there is no reason to let it be external to the article and then deal with the consistency of the relationships between an article and its administrative description.

F INRIA answers I - Sophia

F.1 Creation

F.1.1 What does "digital library" mean in the context of the specific applications you are working with?

For us a digital library is a collection of files containing mathematical theories developed in Coq, mainly with the Pcoq interface, files containing abstract syntax description, and files containing displaying informations, for the mathematical objects.

F.1.2 What is the "content" of your digital library? (A short overview of the data content and format)

The library contains Coq and Pcoq files:

- .v files containing Coq commands (definitions, tactics) - .vo files, containing machine readable Coq objects, obtained by compiling .v files. - .s files, containing abstract syntax trees of Coq commands, obtained by Pcoq. A .s file is obtained typically by parsing a .v file and editing it inside Pcoq.

Additionnaly, in the context of Pcoq, we have pretty-printing files (.ppml files), which contain pretty-printing rules (in the PPML formalism) for displaying Coq objects and Coq commands. In a particular context, we can change the pretty-printer to adapt the display to the current theory.

We have also preferences files that describe, for a particular user, its mathematical notations. They are used to display mathematical formulas with 2D structure, arrows on top of vectors, matrices, etc. In the future these notations will be attached not to a user but to a file (notations change with the theory we study).

Pretty printing rules can be, at least partially, used to produce XML code of formulas.

It is important to note that Pcoq tools and formats (pretty-printing, abstract syntax trees) are almost compatible with XML and Math-ML. This compatiblity will be completed in the future, and ultimately, Pcoq formats will be replaced with XML formats.

F.1.3 How is content added to library? What are the different pathways how this might happen?

.v files are added by using the standard tools given by Coq: compilation of .v files using Makefiles.

.ppml pretty printing files are added using standard editing tools (emacs), and compiled using antlr to produce Java code able to display objects in a box language.

Preferences files are produced dynamically by the user of Pcoq.

F.1.4 Regarding scalability: How many entities does your digital library contain (order of magnitude)? What part of it is or is expected to be equipped with markup?

Our Coq files (.v and .vo) is rather large (several hundreds of files).

Pretty-printing files contain thousands of rules, dispatched in dozens of files.

Preference files are smaller, containing informations about dozens of operators.

F.2 Archiving

F.2.1 What is stored (physically)? Is there information which is generated on the fly on query/retrieval?

.v files are produced interactively by the user.

.vo files are produced by the Coq compilation command (coqc.). They can be loaded dinamically to develop a theory using previous developped theories.

.s files (containing abstract syntax trees of Coq objects) are produced automatically by during the use of the Pcoq interface (in fact they are mainly used as backup files).

F.2.2 Describe under what circumstances content is being removed/treated as temporary/becomes obsolete.

With new versions of Coq, .v files can be obsolete, by mainly they remain correct. this is not the case of .vo files that must be completely re-computed.

Pcoq pretty printing files are also affected with versions of Coq, because of changes in the abstract syntax of Coq, and also because of changes in the concete syntax.

F.2.3 Do you need/use a versioning system?

We use CVS.

F.2.4 Are there interrelated entities in the database (i.e. entities whose meaning depend on a context which cannot be automatically reconstructed from the entity alone)?

All Coq files (.v and .vo) are strongly interrelated, in a loopless manner, as mathematical theories are.

Syntax and pretty-printing files are much more independent. They can be compiled and used independently (even if two rules of two different files can overlap, and then the result can be unexpected, even if it is deterministic).

F.2.5 Is content being stored at multiple physical locations? If not, would you perceive a need for a distributed archive of your data?

Content is mainly stored in two directories in the local network: the installation directory of Coq and Pcoq, and the user working directory. Other directories can be used, by adding Paths in the list of known paths.

F.2.6 What are the consistency constraints?

Consistencies constraints are given by Coq: all .vo files have to be compiled with the same version of Coq. Additionnaly, the dependencies between theories have to be respected during the order of compilation. We use a Makefile with a .depend file (with a tool given by Coq: coq-domakefile).

F.2.7 Is the architecture centralized?

Yes. Both the installation of Coq and Pcoq are centralized. But files containing theories can be dispatched on several locations/machines. Syntax and pretty-printing files have to be also centralized. preferences files are located in the HOME directo of the user.

G INRIA answers II - Rocquencourt

G.1 Creation

G.1.1 What does "digital library" mean in the context of the specific applications you are working with?

For our site, the digital library is a library of theories or results formalised in the Coq proof assistant. Theories include definitions, lemmas and theorems, and proofs of these lemmas and theorems. Theories are mainly of mathematical nature, e.g. about algebraic structures, reals, complex numbers, trigonometry, categories, sets or of computer science nature, e.g. theories about unification, lists, maps, floating-point numbers, cryptography. Similarly for end-results which include the fundamental theorem of algebra, the 4-colour theorem, but also confluence of λ-calculus, correctness proofs of various programs or decision procedures, ...

Definitions and statements are expressed in the logical formalism called the Calculus of Inductive Constructions. For proofs there is currently two representations: as a script of tactic or as a proof-term.

Both representation are, at the time being, unsatisfactory. The scripts of tactics is the more compact representation but it cannot be understand from a human reader because too much contextual information is missing. The proof-term representation is generally very big with all the formal details needed to ensure the correctness of the proof. Therefore proof terms are also difficulty usable by a human.

The database of Coq theories on our server at Rocquencourt provides only the tactic scripts. Bologna has its own database which provides the proof-terms written in natural language (but still with too much details to be easily grasped by a human).

The mathematical objects involved in the library are expressions of the Calculus of Inductive Constructions, declarations (definitions, theorems, axioms) and theories (groups of declarations constituting a module).

G.1.2 What is the "content" of your digital library? (A short overview of the data content and format)

We actually have two databases. The first one describes the so-called standard library of Coq which is provided by default with the Coq system. This library is made from the Coq source files (which include comments in either HTML or TEX format) + an index of the names of objects occurring in the library (names of the notions defined, of the axioms, of the theorems). This database, automatically translated from Coq syntax to HTML format, is browsable and searchable by names. Remark that HTML files are produced asynchronously, not dynamically.

The second database is a collection of theories and results in Coq that users contribute. Users contributions come with an independent description file which contains metadata informations: title, authors, institution and keywords. Keywords are searchable by a CGI script but the files of the contributions are not browsable. Remark that we have small theories (one or two source files) and very large ones (until one hundred files).

Bologna has its own library made from the Coq "compiled" files (both the standard library and the user-contributed library). As a consequence, they get a precise and absolute representation of proofs but (at the current stage of our collaboration) lose the comments put by the developer of the theory in the source file and lose also some syntactic short-cuts or

implicit information that a source file allows. The user contributions are browsable on the Bologna site.

G.1.3 How is content added to library? What are the different pathways how this might happen?

The content is before all a formal content in the Calculus of Inductive Constructions that can be processed, transformed, rendered and indexed by automatic tools. Then come metadata informations and various (non formal) comments that only contributors can provide. Some conventions already exist for these metadatas but it is very ad hoc and rudimentary.

Users submit contributions using a web interface or just by mail.

G.1.4 Regarding scalability: How many entities does your digital library contain (order of magnitude)? What part of it is or is expected to be equipped with markup?

We have 80 user contributions and a overall size of 3 Mb for compressed source files. Since the library content is formal, markup can be built automatically.

G.2 Archiving

G.2.1 What is stored (physically)? Is there information which is generated on the fly on query/retrieval?

For the standard library of Coq, source files are translated to HTML files and stored as HTML files. So is the index of names.

For the user contributions, only a tar-ball of the contribution + a description file are physically stored. Only results of search by keywords are generated on the fly.

G.2.2 Describe under what circumstances content is being removed/treated as temporary/becomes obsolete.

User are free to update their contribution at any time. The only constraint we have is that the contribution is checkable by the current version of Coq.

Usually, new versions of Coq force to update the contribution. This updating work is supported by our team. We keep a copy of each user contribution for all versions of Coq for which a compatible version exists. The default when looking for a contribution is to link to the version of the contribution compatible with the current version of Coq.

We have no committee deciding of the relevance of a contribution. This is why we have small ones and very large ones. We also have some redundancies (e.g. several developments of the algebraic hierarchy).

G.2.3 Do you need/use a versioning system?

We use the Concurrent Versioning System (CVS) for maintaining a revision history of the user contributions.

G.2.4 Are there interrelated entities in the database (i.e.entities whose meaning depend on a context which cannot be automatically reconstructed from the entity alone)?

Yes, some user contributions depend on other contributions provided independently by other users.

G.2.5 Is content being stored at multiple physical locations? If not, would you perceive a need for a distributed archive of your data?

Both Bologna and Rocquencourt have a copy of the public developments in Coq (standard library and user contributions) but not stored in the same format.

Currently, user contributions are submitted to our team. We could imagine that users just give us a pointer to their own repository of their contribution. But I'm afraid the consistency of the database would then be unsure (dead links, incompatible versions). Mirroring can be considered.

G.2.6 If so, what is the distributed architecture?

G.2.7 What are the consistency constraints?

The main consistency constraint is the compatibility of contributions with the version of Coq it is assumed to work with and with the other contributions it possibly depends on. This consistency constraint is about both the mathematical content (assumed to be re-checkable at any time by Coq) and the interdependence of references between contributions or within the standard library.

It should be clear that the consistency can only be ensured for a centralised repository, or at most between cooperating distributed sites. For instance, if an individual user publishes on his site a contribution depending on the contribution of another independent user, nothing can ensure that the http links, the names of mathematical objects or the content of the mathematical objects are not broken or incompatible.

To ensure the consistency of our user contribution web repository, we forbid updating of single files independently. The site is updated as a whole after Coq has checked the mathematical correctness and the logical interdependencies of the whole set of contribution.

G.2.8 Is the architecture centralized?

Yes

G.2.9 Are there "trusted" repositories?

For us, "trusted" means checkable in Coq.

G.3 Retrieval

G.3.1 Please specify in detail the kinds of queries the API (see subsection 3.4. of preliminary requirement analysis) should be able to deal with for your domain of the application.

We expect to allow queries on the informal "meaning" of some parts the library. E.g. "show me all what is about commutative groups", "show me all about trigonometric functions", "has the fundamental theorem of algebra already been formally proved", "is associativity property for lists concatenation already proved", "is there a formal definition of equivalence relations in the library"?

And we expect to allow queries on the formal "meaning" of the objects in the library. E.g. "find a theorem of statement $\forall A.\forall l_1, l_2, l_3 \in list(A), (l_1@l_2)@l_3 = l_1(l_2@l_3)$", "find all equalities involving a pattern of the form $(n_1 + n_2) * n_3$", "find all lemmas stating that a relation is an equivalence relation" (based on the formal meaning of the notion of equivalence in the library), "find all statements which deduces something from being a symmetric relation" (in the formal meaning of symmetric). Ideally we'd also like to look for formal content up to isomorphism (e.g. that statements $A \wedge B$ and $B \wedge A$ are identified).

Query on the informal "meaning" of the library This kind of query relies on meta-logical content. It can look for instance on the name of a notion in the library. E.g. if the formal notion of equivalence in the library has name `equivalence` then a search on the string *equivalence* will succeed.

But we need also more "informal" content in the library. The formal name of the "fundamental theorem of algebra" in the library cannot be these 4 words sentences (at least in Coq where spaces are not allowed in formal names). Is it `FTA`, `FundamentalTheoremAlgebra`, ... or perhaps just `main_theorem`?

Similarly, how to find the statement on "left associativity of addition on natural numbers". Such a long sentence is not what the formal part of the library is able to carry (and actually, the proof of this statement is called `plus_assoc_l` in Coq).

We then definitely need to have a way to relate formal names and their intended meaning in natural language. A first approach is to use canonical name. For instance an associativity property will always be represented by the string `assoc`, left associativity by the string `assoc_l`, addition on natural numbers `Nplus`, etc. But then what to do with less common notions. What could be a canonical name for the Fundamental Theorem of Algebra (knowing that in some countries it is not at all called fundamental: e.g. in France its name is "d'Alembert theorem").

My feeling is then that canonical name are not enough. We need to associate strictly informal meaning (e.g. keywords or sentences in natural language) to formal notions.

May content dictionaries be good for this?

Query on the formal "meaning" of the library As soon as the formal representation of an informal notion has been identified in the library, users should be able to find various informations about this notion in the library: to which other notion it is equivalent? which theorems are about this notion? If the notion is a property, we could ask which theorem states this property or which theorem deduces other informations from this property. This is especially useful for a user itself contributing the library. E.g. he may want to know if a

statement he needs has already been proved formally in the library. Also, if he is currently working on a formal proof, he may ask for all lemmas applicable at the current stage of the proof.

Then we also need to search for theorems by its formal statement. A tool to look for statement up to isomorphism (e.g. $A \to B \to C$ is equivalent to $A \wedge B \to C$) has already been designed but it remains limited. Another tool to look for patterns of statement has also been proved useful.

Finally, another application of search is statistics: how many notions relies on another one. What is the dependency graph of a given theorem. This may require both kind of queries (formal and informal contents).

G.3.2 Please specify what's going to be retrieved.

Retrieval is basically retrieval of the precise content a definition or of a theorem, but then implicitly of the whole file or the whole contribution in which this is defined or proved.

G.3.3 How should the results be presented (HTML page, math code ...)?

In any format, as standard as possible, as fast as possible and as pretty as possible.

G.4 Implementation

G.4.1 How is search metadata being stored in the library and associated with the documents?

For the library of user contributions, the metadatas are in a description file (text format with fields name ended by colon). In addition there is hand-made classification of the contribution.

For the standard library of Coq, the metadatas consist only in an index of names of the mathematical notions defined. This index is produced automatically from the sources. It is stored in HTML format. Remark that comments in the source files can embed HTML tags (and actually TEX also) which are interpreted in the browsable HTML copy of the source file (in the style of javadoc).