INFORMATION SOCIETY TECHNOLOGIES
(IST)
PROGRAMME

Project IST-2001-33562 MoWGLI

Deliverable n. D5.b
Advanced MoWGLI Prototype (distribution)

Main Authors:
Claudio Sacerdoti Coen

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 Overview

The Advanced MoWGLI Prototype is an enhancement of the First MoWGLI Prototype meant to integrate the new functionalities developed in the other MoWGLI prototypes. In particular, in the MoWGLI original release plan it was meant to integrate:

1. distribution management according to the distribution module design developed in the previous deliverables;
2. the extended prototype of the LaTeX-based authoring tool.

Nevertheless, the following events have changed the focus of this prototype:

1. In the previous deliverables of the Distribution working package we already implemented tools for browsing, rendering and consultation of theories. A MoWGLI *theory* is view over a fragment of the MoWGLI distributed library. Concretely, it is an XML document (e.g. an XHTML or DocBook document) that embeds MoWGLI references to objects in the library. Rendering a theory means embedding the rendering of the objects into the normal rendering of the file.

   We did not develop any tool to automatically generate theories. A few hand written theories were provided for debugging, and many more are expected from the D6.a "Validation 1: education" deliverable. To avoid late bug reports for theories, we decided to implement an automatic translator from Coq development files to MoWGLI theories. The tool allows early testing and debugging of theories, and it is surely interesting for the whole Coq user base, completing the MoWGLI exportation and browsing tools.

2. We have implemented and tested a Web service for distribution that fulfills all the requirements initially identified for MoWGLI. A preliminary version of the Web service was already used in the First MoWGLI Prototype. The Web service, called *Getter*, creates a logical, hierarchical abstraction for a physically distributed library, behaving as a sort of proxy and decoupling a MoWGLI URI (that becomes a Uniform Resource Name) from the URLs of the document copies identified by the URI.

   According to the original plan, versioning is not part of the distribution model implemented by the *Getter*. The original idea was that it would be possible to implement versioning on top of the underlying distribution level as a future extension of MoWGLI. However, a careful study by the MoWGLI partners has shown that the implementation of versioning as a separate module on top of distribution is more difficult than expected. In particular, the layered approach would prevent an effective implementation of several interesting functionalities that are required by collaborative development.

   Thus, the MoWGLI partners have developed a change management system based on semantic difference computation that would form the basis of a distributed collaborative development system for content-based mathematics. This architecture is described in Deliverable D5.1 and goes significantly beyond what was anticipated in the MoWGLI grant proposal and in itself represents a significant result of the MoWGLI project.

   Prototypes of selected parts of the new architecture have been implemented (the XML-CVS server and database back-end, a semantic difference computation engine, an XML-diff/patch/merge client), but they are not mature enough to be smoothly integrated in the Advanced MoWGLI Prototype. In particular, the practical work on these components has revealed problems in the underlying technologies we build on, further delaying progress on the prototype. For instance the XUpdate standard for XML-based differencing scripts is underspecified and the existing implementations are largely incompatible.

For the previous reasons we plan to integrate the new proposed distribution architecture only in the Final MoWGLI Prototype (deliverable D.6.d), once the developed prototypes reach maturity. In the meantime, the Advanced MoWGLI Prototype integrates the simpler, Getter based, architecture thus fulfilling the promises in the MoWGLI proposal. The extended prototype of the LaTeX-based authoring tool has been integrated as expected.

Detailed descriptions of the modules that compose the prototype and that have been developed as separate prototypes can be found in the corresponding deliverables.

2 Introduction

The MoWGLI project aims at exploiting the possibilities offered by a content-oriented, machine understandable encoding of mathematical knowledge in the creation of a digital library of mathematics. Currently, in accordance with the timetable of the workplan, all the functionalities expected in the final version of the prototype have been implemented as prototypes that will be refined during the evaluation period.

The library comprises documents from the Coq standard library as well as from several other Coq contributions, and TeX documents that have been processes with the LaTeX-to-MathML converter tool (deliverable D4.d). These two kinds of documents (formal mathematical proofs and TeX articles) have been chosen because they represent two opposite categories: highly structured, automatically generated pieces of mathematical knowledge and handwritten scientific articles in a widespread typesetting markup language.

The Advanced MoWGLI Prototype replaces the First MoWGLI Prototype and is accessible at the following URL:

`http://mowgli.cs.unibo.it/library/`

Starting from this page, access is provided to subsections for:

1. Browsing and searching the content of the library
2. Augmenting the content of the library
3. Managing the content of the library

In the following sections we briefly summarize the functionalities offered by the prototype, and we provide a few insights on the Getter Web service that handles distribution. Detailed instructions on the usage of the prototype can be found directly on the Web site.

3 Browsing and searching

Both the library of formal mathematics exported from Coq and the library of papers extracted from TeX articles can be browsed. For the former library more functionalities are provided, since much more information is available in the original documents. Thus, in the following description, we will focus on the library of formal mathematics.

For each object in the library (i.e. each lemma, definition and so on) it is possible to:

- Retrieve the file in its semantic form, i.e. as an XML-encoded lambda-term. Useful as a back-end for Web services that provide further functionalities and that require knowledge about the formal semantics of the object. This functionality is directly provided by the MoWGLI Getter Web service.

- Compute and retrieve the OMDoc content description of the object. MathML Content is used to describe formulae inside the OMDoc document. Useful as a back-end for Web services that provide further functionalities and that understand the OMDoc proposed standard format for content level representation of mathematical developments. This functionality is provided by the combination of the MoWGLI UWOBO Web service — used to apply XSLT stylesheets — that interacts with the MoWGLI Getter.

- Compute and retrieve a description of the object in XHTML. Formulae can be described either in MathML Presentation or in XHTML (if a loss of rendering quality is acceptable). The latter option is useful for browsers that are not fully MathML compliant. This functionality is also provided by the orchestration of the MoWGLI Getter and UWOBO Web services.

- Explore the structure of the proof by folding/unfolding subproofs and following hyperlinks to other proofs and definitions. This feature is only available within JavaScript enabled browsers.

- Understand the logical structure of a formal library by inspecting the mutual dependencies between objects shown as graphs of dependencies. This functionality is provided by the orchestration of several MoWGLI Web services: the Getter (to retrieve the documents); UWOBO (to apply the XSLT stylesheets that computes the dependencies); a Web service that wraps a library to draw a graph given its textual description; an ad-hoc Web service that implements the data structures used during the computation of the dependency graph.

- Check the correctness of proofs and definitions. The functionality is provided by a Web service developed outside MoWGLI by the HELM project. The Web service interacts with the Getter to retrieve the documents.

- Show the metadata associated to the object. Metadata can either be stored in an RDF file managed by the Getter or they can be stored in a relational database (tested with PostgreSQL or MySQL) and transformed on the fly to RDF files by yet another Web service.

A Web interface to a search engine also developed in MoWGLI is provided to the user. The search engine is implemented as a Web service, and it is tightly integrated with the rest of the interface. In particular, the answer pages returned are MoWGLI theories that list the URIs of all the objects found. The theories are immediately processed by the UWOBO Web service, that retrieves each object in the page by contacting the Getter and that process it and embeds its rendering inside the page. The stylesheet applied by UWOBO also adds hyperlinks from the rendering of the object to the MoWGLI page that shows the object alone, together with the buttons that provide all the browsing functionalities described above.

The kind of queries available in the search engine interface are those made possible by the MoWGLI set of metadata already described in the deliverable D3.b "Metadata model". The metadata are currently automatically computed from the XML files thanks to the tool described in the deliverable D2.g "Tools for automatic extraction of metadata".

4 Augmenting the content of the library

The user can either contribute a Coq development or a TEX article. The XML files contributed can either be generated by the users on their own machines — if the MoWGLI tools have been locally installed — or they can be generated by scripts running on the MoWGLI server. In the latter case, the user just uploads to the MoWGLI server the Coq or TEX files that must be exported to XML.

In either one of the previous solutions, the XML documents eventually generated are assigned both a logical name (a URI that is a Uniform Resource Name) and a URL (by publishing them on a Web server or an FTP server). The MoWGLI Getter is notified with both the logical and physical name so that, later on, the MoWGLI tools can retrieve the file by using the logical name only: whenever a file is required, a request to the Getter is issued by specifying the file logical name; the Getter locates the file (i.e. its URL), retrieves it and returns it to the client, thus behaving as a Web proxy.

The precise steps to submit a contribution were already described in the report about the First MoWGLI Prototype and they have not been changed. Thorough information for the submission can be found on the MoWGLI pages.

5 Library management

This section of the interface allows to remove a Coq contribution from the library and to update an existing contribution with a new version. The user interface is unchanged with respect to the interface of the First MoWGLI Prototype.

6 The Getter

Before describing the services provided by the Getter and its implementation, let us review briefly the requirements that influenced its implementation.

An important requirement of MoWGLI is to keep a minimum burden on the user: no client-side software should be required to consult the library and as little as possible should be required to interact with it. In the same way, no client-side software should be required to contribute to the library by publishing a new result.[1] In particular, we want any user with a Web space (either HTTP or FTP) to be able to publish a document/object without having to install any particular software. The main reason is that the Web space of a user is often hosted by a provider that does not allow new software to be run. Moreover, the simple HTTP publishing model has already proved highly effective in creating fully distributed hypertextual

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

libraries of knowledge, which we see as a motivation against a more centralized solution, such as a net of cooperating databases to which the user can submit a contribution (the solution adopted in MathWeb[2]).

Our distribution model has been designed under the assumption of immutability of the documents. In fact, the original idea was to consider each version of a document as a different document, and to add later on a versioning layer on top of distribution. The main motivation was granting consistency of the document base: a new release of a document would not break any already existent mathematical development, since the old copy would still be the copy referenced by other documents. The immutability assumption will be dropped in the forthcoming new MoWGLI distribution architecture, that implements a proper versioning model.

Since for now documents are immutable and are already identified by logical names (URIs) rather than physical names (URLs), many copies of them can be stored on different servers, and users are free to retrieve them from the nearest or less loaded server, achieving load balancing.

Finally, users cannot be forced to retain a copy of their documents forever, even if other documents refer to them. By allowing different copies on different servers, though, we give users who are interested in a contribution the possibility of making a local copy of it and starting its distribution. This way, interesting documents tend to augment the number of their instances, avoiding the danger of disappearing, while uninteresting ones could simply stop wasting space, being deleted.

We leave open the problem of choosing a naming policy that prevents independent users from choosing the same URI for different documents, thus creating name clashes. To face the issue, one solution is a centralized naming authority. A simpler solution could just be considering the URIs as completely meaningless identifiers, leaving to the high level tools the problem of organizing the library by means of metadata. In such case, we could easily adopt a naming scheme that avoids the problem of name clashes by construction, as the one adopted for package names for the Java programming language.

Having illustrated the requirements, we can go on with the description of the implementation, skipping the uninteresting details.

The Getter is a Web service whose main task is mapping an URI to the relative URL and then retrieving the identified document, that is returned to the user. According to the similarities with package management systems, the implementation has been largely inspired by the APT package manager.[3]

In more details, the main method of the Getter takes an URI and returns a copy of the document identified by the URI, downloaded from a distribution site that provides it. In order to know which documents a server provides, each server publishes a list of the URIs of its documents associated with the respective URLs. Users of the Getter, instead, configure the Getter with an ordered list of servers. Periodically, the getter contacts each server in the list and retrieves the list of documents the server knows, along with their associated URIs. The information retrieved is stored in a local table (implemented as a NDBM database) that maps each URI to the URL of the first server in the list providing a copy of the document. The table is consulted to resolve the URI every time the main method of the Getter is invoked.

As Xml files are very verbose, we give users the possibility of storing them in a compressed

[2]`http://www.mathweb.org`

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

form. The Getter is also responsible for decompressing the documents before giving them back to the client. Moreover, the Getter is able to retrieve the files from an HTTP, FTP or NFS server, passing them back to the client using a uniform HTTP server interface. Finally, as the Getter is supposed to reside closer to the user than the distribution server, it implements a caching mechanism that reduces downloading time of already retrieved documents. All these reasons forced us not to use the HTTP redirect method to map URIs to URLs, as done by PURL which is an otherwise similar tool to resolve persistent URLs (URIs, in fact) to URLs.[4]

The Getter also provides other administrative and less interesting methods. Among them there are methods that force the reconstruction of the URI resolution table, methods to notify the Getter about the publication of a new object and methods to check the existence of a document associated to a given URI.

As a final remark, we highlight again the twofold nature of the Getter: on one side it is a Web service that can be contacted by HTTP clients, thanks to the HTTP GET binding; on the other side it is an HTTP client that can download files from a remote HTTP server or that can invoke methods of other Web services that have an HTTP GET binding. This compositional nature is typical of all the Web services developed in MoWGLI, and it constitutes a fundamental ingredient of the architecture of MoWGLI. In particular, in MoWGLI we are developing high level Web services that are implemented by means of HTTP calls to lower level Web services.

[4]PURL (`http://www.purl.org`), being general-purpose, cannot rely on document immutability. Hence, at most one copy of a document can be available and no load balancing is provided.