A Presentation Architecture for Individualized Content

Alberto Gonzáles Palomo, Paul Libbrecht, Carsten Ullrich
University of Saarland and German Research Center for Artificial Intelligence
Saarbrücken, Germany
{alberto,paul,cullrich}@activemath.org

Abstract

A modern approach for generating individualized web-sites is to compose a page out of individual elements, for instance XML-fragments, which is eventually transformed to HTML. If the generated pages differ for each user, then the required transformation processes put a heavy load on the server, hence slowing down response times significantly.

The learning environment ACTIVEMATH uses this composing approach to generate learning courses that suit best the needs and goals of the individual learner. For instance, depending on their current knowledge, different users that learn the same content get presented courses that differ in length and in amount and difficulty of exercises and examples. Currently, the learning materials are transformed in one step from XML to HTML (or other output formats).

The learning materials are encoded in a language called OMDoc which encodes semantically the fragments as well as the mathematical formulae. This article hence provides an approach to the answer "How can Semantic Web technology be used to improve adaptation and information retrieval?". Moreover the ability to generate multiple output formats from the same content source is a central requirement of the architecture. It provides a first step towards device adaptation providing, currently, an on-screen HTML version and a printable PDF version.

This article describes the architecture we developed to solve the performance problems that arouse out of the page generation process. In this architecture, the generation process is divided into several layers, with each layer adding/transforming well-specified data. Among other advantages, this approach allows caching of individual transformed fragments. We hope that in this way the performance problems can be reduced.

1 Motivation

A current trend in the WWW is the generation of individualized web pages. Individualization as meant as in this article covers the complete range from inserting small pieces of text (such as the user's name) to composing the complete text of a page depending on properties of the user. An area where this kind of individualization certainly makes sense is education. Although most of the (commercially and freely) available learning materials are of a static nature (HTML pages, sometimes extended by applets, PDF, PPT, etc.) learning materials individually tailored to the student's needs definitely support their learning process in a favorable way.
One technically advanced e-learning system is ACTIVEMATH [7], a web-based learning environment that provides a throughout individualization. Learning materials, such as courses, are available in a common "static manner", but are also constructed dynamically according to the student's learning goals and his current knowledge. The pedagogical advantages of the dynamic

individualized generation are numerous: The learner has at his hands a course specifically tailored to his needs that contains only the necessary content at the adequate knowledge level. However, when we were using ACTIVEMATH for the first time in an university course, some drawbacks of technical nature arouse. Several complaints considered the sometimes slow response of the system. In the first, ad-hoc implementation, every time the student visited a page, the system (re-)builts this page from scratch. These fetching and transformation processes of course put a significant load on the server.
To overcome this and other problems detailed below, we analyzed and structured the presentation process, and, building on this analysis, developed a general architecture for the generation of individualized web pages. In short, we divided the presentation process in several separated stages, where each stage adds distinct individual information. Thereby, caching is possible in several places.
We think this architecture is of general interest. Although developed in an educational setting, the architecture we propose is general enough to be applied in all settings where throughout individualization is needed.
We will start with an overview on the generation of individualized web pages, how it is done in ACTIVEMATH, and the advantages it offers. Then, after talking about the problems that arouse out of our first, "naive" presentation engine, we will present the new, layered presentation architecture. The article concludes with a comparison of our architecture to existing frameworks.

2 Individualized Web Pages

From the very beginning of the web, first approaches of individualization were realized by CGI scripts which generated the complete text of the delivered page. They were followed by *script-in-page* approaches, such as Java Server Pages or Active Server Pages. There, the HTML-code of a page contains special tags that are replaced by individual content. The advent of structured content provided by XML encodings opened the door to data processors: These engines generate browser-viewable content from a content whose structure is well defined and well known, the eXtensible Stylesheet Language Transformation (XSLT) process is a good example of such an approach: using an XML-encoded source and a few parameters, it produces the viewable content.
A good example of state-of-the-art individualized generation of web pages based on an underlying XML-representation is ACTIVEMATH. In the following section we will provide some details on ACTIVEMATH with respect to this feature.

3 ActiveMath, a Web-Based Learning Environment

ACTIVEMATH is a web-based, user adaptive environment for mathematics education. It employs content encoded in a semantic XML-representation and integrates several mathematical systems to support exploratory learning.
ACTIVEMATH generates individual courses for each learner. An example is shown in Figure 1. In the example, both users want to learn about the mathematical concept *morphism of groups*. Eva, on the left hand site, already knows a bit about the mathematics necessary to understand morphisms. Therefore, the course generator of ACTIVEMATH has composed a shorter course for Eva (see her table of content) than for Anton, on the right hand site, who misses some prerequisite knowledge. In addition, the content within a page differs. Eva know the concept *monoids* quite well, and the course generator generated a page that serves to reactivate this knowledge. Anton however, for whom this concept is unknown, gets presented a page that among others provides a detailed example on monoids.
More generally, ACTIVEMATH provides individualization with respect to a complete course, the elements on a page within a course, and the text within the elements. Hence, ACTIVEMATH requires a powerful underlying knowledge representation. The following section provides some details about it (for more information, see [9]).

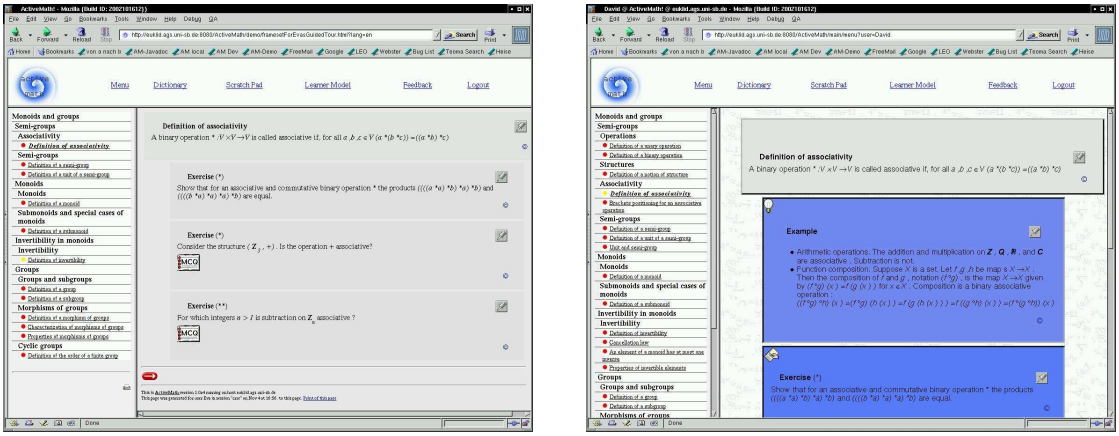

Figure 1: Two courses for the same learning goal, but for different learners.

3.1 ActiveMath's Knowledge Representation

ACTIVEMATH knowledge representation is a semantic XML encoding. As ACTIVEMATH is currently targeted at teaching math, it uses `OMDoc`. `OMDoc`, documented in [6], provides means to represent the structure of a mathematical document. It allows the encoding of different categories of items at paragraph level: mathematical concepts such as definitions and theorems, and further items such as examples, exercises, and elaborative texts.

`OMDoc` allows the annotation of items and documents with metadata. The metadata covers pedagogical aspects of an item such as `difficulty` and relations between the items such as `depends-on` or `similar-to`.

The knowledge representation contains some but not all metadata of the Learning Objects Metadata standard of IEEE [5] but includes additional relationships and characterizations that are missing in this standard.

Furthermore, `OMDoc` allows to represent the semantic of a mathematical formula. Whereas in an usual text-document $a + b$ simply represents a string and $+$ the ASCII character 43, the `OMDoc` encoding clearly specifies which mathematical symbol is meant. For instance, besides the addition of numbers, $+$ often denotes a group operation. To know the semantic of a mathematical object is especially important if the content is used in or interchanged between external systems such as computer algebra systems (which ACTIVEMATH uses for interactive exercises). Another example are copy and paste operations where the user marks a formula and wants to copy the corresponding mathematical object, not its textual representation.

3.2 Current Generation and Presentation Process

Mathematical learning material, i.e., the course and its pages, is dynamically generated on the user's demand by selecting and composing the `OMDoc` items. The course generation process happens as follows:

Starting from the goal concepts chosen by the user or a teacher, all concepts they depend upon are retrieved recursively from a knowledge base. The result is a collection of all concepts that need to be known by the learner in order to be able to understand the goal concepts together with additional material such as examples and exercises. Then pedagogical rules are applied which take information about the user (stored in an user model) into account. These rules determine, e.g., the types of items to appear on a page, the appropriate level of difficulty and the order in which the material is presented. The result depends on the available content, the current state of the user model, and on the chosen *learning scenario*: Different documents are generated for different scenarios such as "overview" or "exam preparation". Eventually, the learning materials are ordered and put into a hierarchy.

For the presentation, the generated `OMDoc` pages are transformed via XSLT to HTML, PDF, or SVG. A typical delivery of an HTML page is triggered by an HTTP request containing the user name, page number and an identifier of the current course. The ids of the to be presented `OMDoc` items are looked up using the page number and the table of content of the course. Then they are retrieved from the knowledge base. The resulting XML-document is pre-processed (adding of server specific information, see Section 4.2) and is then transformed via XSLT which directly produces the HTML page annotated with additional individual information (i.e., allowing the user to resume a course at the position he left). PDF-documents are produced in a similar way; additionally the PDFLATEX[1] program is called.

3.3 Advantages of Dynamic and Individualized Generation of Web Pages

The dynamic and individualized generation of web pages realized in ACTIVEMATH offers several advantages:

Re-use of Content Developing content, especially learning materials, is time and money consuming. By re-using and improving already created content, (hopefully) the costs are reduced and quality enhanced. ACTIVEMATH offers a very fine-grained re-use: Instead of complete pages (or even courses), single paragraphs form the basis of the re-use. This level is not only suited for learning, for instance, parts of a *news story* can be inserted or skipped depending on the background knowledge of the reader. In this way, different parts of the content can be re-used for a variety of purposes.

Multiple Output Formats Generating the content from a semantic, presentation-independent knowledge representation makes it possible to render to multiple output formats depending on the current needs of the user. Not everything that looks nice at the screen still looks nice when printed out. Guidelines for ergonomic layout differ depending on whether the content is targeted for print or online viewing. Therefore, a system should provide the possibility for elements to have different renderings depending on the output format. But this is only possible if the concrete rendering is not hard-coded in the content. For instance an `emphasize` is not equivalent to a purely presentational `bold`. The style-sheets decide which presentation is chosen for emphasize.

Personalization of Content As already mentioned, ACTIVEMATH offers truly individualized content generation. The advantages of such individual generation are numerous: The user needs less time to find and to learn about the content he is interested in, he is not de-motivated by content too difficult for him to understand, neither bored by facts he already knows about.

Combining Learning Materials The author, who writes the content, and the editor (in an educational setting the teacher), who selects which content to present, can but do not have to be the same person. Therefore, combining learning materials can be far more laborious than on first sight, especially in mathematics and other formal sciences: Different authors tend to use different notations. A well known example is the logical concept of *implication* which is presented differently in almost every book about logic, e.g., as $\Rightarrow$ or $\supset$.
The concept of *composition* is a more challenging example that requires not just replacing a symbol: To denote the composition of g followed by f, in general, a German author would write $f \circ g$, whereas an English author would write gf.

[1] LaTeXis a typesetting program designed for high-quality composition of content. PDFLATEX is a variant that produces PDF.

Combining such materials when they are written in traditional presentation-oriented languages requires tedious manual rewriting. However, if a presentation-independent knowledge representation is used, then the is presentation specified separately from the content, for instance in a XSLT-rule. To determine their preferred presentation, authors that combine materials from different sources simply specify the presentation rules for the necessary concepts; they do not have to worry changing the content itself.
We esteemed the background information given in this section necessary specify the setting of our system and we omitted several technical details concerning semantic encoding. For a more concise overview on knowledge representation and management, please see [9].

3.4 Problems Regarding Presentation

Our first "naive" presentation engine that generates the desired output format directly annotated with individual data in one step gave rise to several problems:

Server Load At every page request the presentation process was repeated completely starting from fetching the content to transforming it to the output format. Of course, this puts a heavy load on the server. Especially the transformation process requires a lot of resources. Caching the fully generated pages does not alleviate the problem neither, as although the students following a course would share some of the items on a page, the other items such as exercises and examples will certainly differ.

Performance As a direct consequence of the high server load the response time of the system decreased, in particular if a large amount of users accessed it simultaneously. This is especially unpleasant as for web applications slow response times are critical. For instance, our students reported that the long delays were a major source of de-motivation.

Fixed Presentations If several presentations for the same concept are possible, some instance has to decide which presentation to use. For students in a course this decision is normally taken by the teacher as he wants to assure that all his students work with the same notation. An independent learner however, can very well make his own choice. Our former presentation architecture used one stylesheet for all users of one ACTIVEMATH installation, thereby disallowing a flexible choice of the presentation.

Missing Abstractions The recent development of the PDF presentation process raised several missing abstractions in our current presentation architecture. As it was primarily targeted for HTML output, the assembly process of the `OMDoc` content was optimized with respect to HTML. For instance, multiple choice exercises were inserted in a special way directed at the presentation in a pop-up window (which is of course not suited for a print-version of a page).

4 Layered Presentation Architecture

4.1 Processed Data

We performed an analysis of the data that is processed/added during the presentation process to determine to which extent the process could be optimized with respect to the above problems. The analysis yielded the following kinds:

Content. Obviously, the presented content forms the major and most important type of data. Content is mostly static in the sense that the containing text does not change (but see Section 4.4.2). However, the overall content is of course dynamic, as different users get different content.

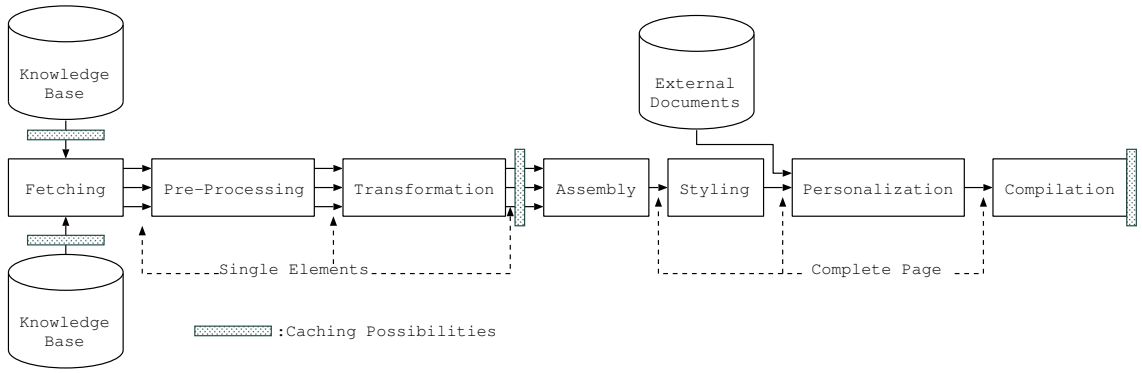

Figure 2: A graphical representation of the layered presentation architecture

Server-Specific Information. This subsumes data added by the current server, such as the version of the used ACTIVEMATH, the address of the server, links, or resource descriptions for interactive exercises.

Presentation Information. This data specifies how specific symbols are to be rendered (represented in XSLT-stylesheets).

Personal Information. While the individual learning materials that are presented to an user are covered by the above content type, personal information is additional individual data added on top of the content. A good example is the indication of the state of knowledge of the user with respect to an element. For instance, if Anton has only very limited knowledge of an element, it can be annotated in a special way, e.g., underlined with red color. Another example for HTML-presentation is the preferred CSS-stylesheet of the user.

4.2 Splitting the Presentation Process

Following the analysis we split the presentation process in several layers, with each layer adding/transforming a specific kind of data. Figure 2 provides a graphical representation of the new architecture. The layers are the following:

Fetching. Collects requested content from the knowledge base. The output of this stage are XML-fragments.

Pre-Processing. Inserts server-specific information into the XML content. For instance, if the content is contained in several distributed knowledge bases, this step changes the ids of the elements to avoid duplicated ids.

Transformation. Performs a first transformation to the desired output format by the application of an XSLT-stylesheet to the document. The output of this stage are HTML or LaTeX-fragments.

Assembly. Joins the fragments together to form the requested pages. This layer uses the XSLT presentation information.

Personalization. Uses personal information to add individually different "beautifications" to the document, such as knowledge indicators, for the HTML generation the user name in the HTTP-links and the used CSS-stylesheet, for LaTeX the used macro-packages.

Compilation. Applies further processing to convert the content presentation into a format that can be displayed by the client. This step is not needed for HTML as HTML can directly be presented by web browsers. However, PDF output requires the compilation of the generated LaTeX-sources.

4.2.1 An Example

This section provides a small example page of the page generation and shows how the content (simplified `OMDoc`-element) is transformed in each stage. Let's assume that the user requests a page that contains only the definition `def1`. In a first step, this content is fetched from the knowledge base:

```
<definition id="def1">
 Definition 1 with a reference to
 <ref xref="def2">definition 2</ref>
</definition>
```

The pre-processing step replaces the ids of the elements. Here, it adds the name of the knowledge base as a prefix to the id, yielding `mbX_def1`.
Then the XML-fragment is transformed to HTML using XSLT:

```
<div class="definition" id="mbX_def1">
 Definition 1 with a reference to
 <a onClick="openInDictionary('mbX_def2')">
  definition 2
 </a>
</div>
```

In the assemble stage, the elements (in this case only one element) are assembled to form a complete HTML page:

```
<html>
 <head/>
 <body>
  <div class="definition" id="mbX_def1">
   Definition 1 with a reference to
   <a onClick="openInDictionary('mbX_def2')">
    definition 2
   </a>
  </div>
 </body>
</html>
```

Eventually personal information is added and the page is send to the user's browser:

```
<html>
 <head>
  <link rel="stylesheet"
        type="text/css" href="colored.css"/>
 </head>
 <body onLoad="initializePageUser('Anton')">
  <div class="definition" id="mbX_def1">
   <img src="lowknowledge.png"/>
   Definition 1 with a reference to
   <a onClick="openInDictionary('mbX_def2')">
    definition 2
   </a>
  </div>
 </body>
</html>
```

4.3 Caching Possibilities

Talking about optimizations only makes sense if the assumptions under which the optimizations take effect are made explicit. ACTIVEMATH was designed with two different use cases in mind: In the group setting a large amount of users access more or less the same content. Although the specific content of the pages can differ, the materials that need to be retrieved from the knowledge base can be specified approximately. In an educational system this would correspond to learners following a lecture, where the content will be the covered domain; in an online news server this content could be the headline articles.
The second use case is the independent self-guided user. No assumptions can be taken about what content he is interested in. For this group, caching the content of one user would not help a second one, as most probably they are interested in different topics.
We designed ACTIVEMATH primarily for being used by a number of students learning about the same content. Hence, the caching we propose is mostly directed at the group case. The second case needs further investigation, although the proposed caches will not have any negative effect for the self-guided use case.
The diagram in Figure 2 indicates three points at which caching can take place:

At Fetching Level. ACTIVEMATH can retrieve its content from different knowledge bases that can be distributed anywhere in the web. As response times will vary heavily, caching once retrieved content at the ACTIVEMATH server can yield better performance, especially for slow connections.

After Transformation. Applying XSLT-transformations requires a lot of resources. Therefore we decided to cache the individual elements after their transformation. A drawback is the additional memory consumption: For every output format, a proper cache is required. In addition, personalization data has to be added after the transformation. For HTML, this can be achieved using another XSLT-transformation and/or JavaScript, for LaTeX, macros are used.

Problems can arise if elements are to be presented differently depending on the other elements on the page, as at the time of transformation, the transforming component does not know which elements will occur on the same page. For instance, a reference to another element can be transformed to a hyperlink that opens a new window with the referenced element if it is not present on the current page. Otherwise, if both elements occur on the same page, the focus of the page is changed to the anchor of the referenced element. However, we were able to overcome this and similar problems we encountered by the use of Javascript.

After Compilation. Caching a completely generated and individualized page speeds up access of often visited pages and going back/forward within a site. This cache is usually provided by the browser cache and can be controlled to some extent using the `http` meta tag `expires` that takes as an argument the date when to refetch the data from the server. However, often this decision can only be taken by the server itself. More often than not, whether a specific page changes does not depend on the time passed since the last visit but on the actions taken by the user. Adding such a cache on the server side definitely adds unrealistic memory requirements if the amount of users is unlimited. In cases that restrict either the amount of users or the number of groups that access different content, caching complete pages can be an option.

4.4 Additional Considerations

4.4.1 External Documents

Another advantage of the layered architecture is the personalization of third-party content not available in the underlying XML-representation.
Most authors don't feel comfortable changing from their preferred content format to a new one, as the change requires new tools, learning how to use the tools and the new format, and, in the case of an non presentation-oriented format, imagined loss of control over layout issues. Furthermore, for old content not to be lost, it requires extensive manual work for conversion.
In principle, performing the personalization after the assembly allows to personalize content not generated by the presentation process itself. These external documents have of course to follow certain conventions, it is certainly not possible to personalize arbitrary documents. For instance, HTML offers manifold ways to obtain a paragraph, e.g. adding line breaks before and after a text block, or including the text within a `div` or `p` tag.
We made the experience that as a first compromise towards completely switching to `OMDoc`, authors sometimes prefer adapting their old content manually with respect to some conventions, e.g., representing a paragraph with the `div` tag, and adding a uniform `id` attribute to them. In this way, some personalization can be performed on this content, e.g., adding knowledge indicators.
Yet one has to keep in mind that this approach is only a compromise. It neither offers the full functionality of personalization nor will it satisfy the author in the long run, especially with respect to re-usability.

4.4.2 Randomized and Generated Content

Some content can not be cached at all because its text is generated on the fly. We distinguish between randomized and generated content:
A good example for randomized content are multiple choice exercises. Every time ACTIVEMATH presents such an exercise, the order in which the possible choices are presented is randomized. As simple as it is, it hampers students to simply copy the answers from their neighbors.
Generated content is content that is generated from an abstract representation. An example are exercises involving statistics. Statistic can be applied to a number of areas, ranging from the probability whether a person falls ill to the probability that some products sells better than others. But the underlying mathematics remain the same. Therefore, an abstract representation can specify the basics of the exercise, and the concrete instantiations are generated with respect to the field of the learner.
These kinds of dynamic content should of course not be cached. Therefore, the assembly process has to be able to know whether to take an item from the cache or whether to request it again from a content-generator (not shown in Figure 2). This can be achieved by adding a non-cache attribute on the elements or by giving the assembly a list of these elements.

4.4.3 Lowering the amount of requests

We also aim at avoiding the load of a document for the sole purpose of performing small, local updates, for instance, changing the colored bullets in the table of content that indicate the mastery value of the user for the topics of the page. These updates are performed via a JavaScript connection to the server. This approach can be generalized to a message-based communication, making browser components *agents* communicating to the components of the server, thereby offering much more flexibility and reducing the amount of information traveling between the brower and the server.

5 Related Work

A huge quantity of systems in education offer individualization of content, see, for instance, the recent proceedings of the International Conference on Intelligent Tutoring Systems [3] or the proceedings of the Conference on Adaptive Hypermedia [1]. Brusilovsky [2] provides a comprehensive overview on adaptive hypermedia techniques, however the techniques he mentions focus on manipulating pre-made pages, for example hiding/showing a paragraph of text, enabling/disabling hyperlinks, or changing the sequence in which the pages are presented to the user.

Recent systems that provide the possibility of assembling pages and courses from smaller *learning objects* depending on user properties are [4] or [8]. Sadly, they do not provide details regarding the technical aspects focused in this article.

A very powerful presentation architecture is the Cocoon Publishing Framework[2]. It offers stylesheet processing added with XML-creation and caching at all levels. The Cocoon framework is very flexible, but consequently relatively complicated. Furthermore, it is hard to debug because of its purely stream-based processing (streams being either byte-streams or, most frequently, XML-parse-tree events). In comparison, our presentation architecture based on an in-memory XML-representation provides more expressive accessors and manipulations for the special case of our `OMDoc` encoding. This supports authors and developers to detect and resolve presentation errors.

6 Conclusion and Further Work

We proposed a layered presentation architecture that divides the page generation in several stages, where each stage adds distinct information. Thereby, more elaborate caching strategies are possible than in an one-step generation. We think our approach is especially helpful if more elaborate individualization takes place than simply inserting a user name. In scenarios where the content presented to the user is composed of parts that are retrieved from a knowledge base, depending on individual properties, the layered architecture can very well be applied and lead to noticeable performance increases.

The implementation of the layered presentation architecture is currently underway. We will then conduct an exhaustive analysis of the performance. In particular we are interested to what extent performances increases under realistic conditions, i.e., in a course, by transforming the single elements for themselves and only later composing them to form complete pages compared to composing the complete page and then transforming it.

Furthermore, we will investigate the effects of the different cache positions on speed increase. Preliminary experimental results show that, in the HTML case, the delivery of a page whose items are all cached after transformation is at least 100 times faster as it only involves merging byte-streams. Nevertheless, exact data can only be gained under real world conditions.

The architecture is being implemented within the ACTIVEMATH system. ACTIVEMATH was developed with modularity and openness in mind so that its components can be easily reused. It is open source and available free of charge in a non-commercial setting[3].

[2] `http://xml.apache.org/cocoon/`
[3] `http://www.activemath.org`

References

[1] P. D. Bra, P. Brusilovsky, and R. Conejo, editors. volume 2347 of *LNCS*. Springer-Verlag, 2002.

[2] P. Brusilovsky. Adaptive and intelligent technologies for web-based education. *Künstliche Intelligenz*, 4:19–25, 1999.

[3] S. Cerri, G. Gouarderes, and F. Paraguacu, editors. volume 2363 of *LNCS*. Springer-Verlag, 2002.

[4] O. Conlan, V. Wade, C. Bruen, and M. Gargan. Multi-model, metadata driven approach to adaptive hypermedia services for personalized elearning. In P. D. Bra, P. Brusilovsky, and R. Conejo, editors, *Proceedings of the second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 2347 of *LNCS*, pages 100–111. Springer-Verlag, 2002.

[5] IEEE Learning Technology Standards Committee. Learning objects metadata, 1999. Available from `http://www.edna.edu.au/edna/aboutedna/metadata/analysis/LOM.htm`, see also `http://ltsc.ieee.org/`.

[6] M. Kohlhase. `OMDoc`: Towards an OPENMATH representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. See also `http://www.mathweb.org/omdoc`.

[7] E. Melis, E. Andres, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Activemath: System description. In J. D. Moore, C. Redfield, and W. L. Johnson, editors, *Artificial Intelligence in Education*, pages 580–582, Amsterdam, 2001. IOS Press.

[8] M. Specht, M. Kravcik, R. Klemke, L. Pesin, and R. Hüttenhain. Adaptive learning environment for teaching and learning in WINDS. In P. D. Bra, P. Brusilovsky, and R. Conejo, editors, *Proceedings of the second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 2347 of *LNCS*, pages 572–575. Springer-Verlag, 2002.

[9] The ActiveMath Group. Knowledge representation and management in ACTIVEMATH. To appear in the Proceedings of MKM'01 in Annals of Mathematics and Artificial Intelligence, 2003.