The Technology of Mumie-JAPS

Tilman Rassy <rassy@math.tu-berlin.de>
Ruedi Seiler <seiler@math.tu-berlin.de>

September 26, 2006

$Id: techn_mumie_japs.tex,v 1.1 2006/07/24 09:16:19 rassy Exp $

1 The Learning Environment ”Mumie”

”Mumie” is a e-learning platform optimized for experiencing, learning and teaching mathematics in a multimedia environment going far beyond simple document handling. Content in the form of fine granular, reusable and interactive elements are organized in math-conform ontologies and can be put and brought to action in different learning scenarios. Visualisation of scientific relations in the form of science nets makes possible non-linear navigatin in the ”Legoland of Mathematical Building Blocks”.

2 Design principles

The design of the Java application server Mumie-JAPS is based on the following principles:

- **Strict separation of content and presentation:** Content contains no layout information and is completely independent of the context in which it is rendered.
- **Dynamical creation of web pages:** Almost any web material, including (X)HTML pages, CSS stylesheets, XSL stylesheets, and JavaScript files, are created dynamically. This allows for a maximum of user adaptivity. To some extent, this is possible with binary contents as, for instance, pictures and applets, too, since JAPS can choose between different ‘realisations’ for the same ‘generic’ document depending on the context.
- **XML technology:** Non-binary content is stored as XML in the database. This is true not only for (X)HTML pages, but also for CSS stylesheets, XSL stylesheets, JavaScript files, and any other text based content. This allows for a unique way to process those contents.
- **Theme concept:** Each user can choose a so-called *theme*. The theme controls the layout of documents, i.e., colors, fonts, etc., but also mathematical flavours, e.g., whether vectors are displayed as bold symbols or with an arrow on top. The concept is realized by means of ‘generic’ and ‘real’ documents: A generic document plus the theme is mapped to a ‘real’ document. Thus, generic documents are placeholders for real ones, with a different implementation for each theme.

- **Reliable referencing:** If a document references another one (as a link, an image, etc.), this is done by a so-called *local id (lid)*. The lid plus the id of the referencing document is mapped to the id of the referenced document by a special database table. This has several advantages: First of all, we avoid hard-coded database id's in the content, avoid redundance in the database, and can update the referenced document without touching the content. Furtheremore, if the documents have not yet been checked-in to the server, filenames rather then database id's may be assigned to the lid's, which facilitates the authoring process a lot: Content can be written without knowing the database id's in advance, and previews can be created as static pages.

- **Version control:** JAPS has a build-in version control system. If a document is checked-in for a second time, the former version it not replaced but saved. Thus, it is always possible to recover older versions. Per default, all references pointing to a document are updated if a new version of the document is checked-in. This is possible due to the 'reliable referencing' concept described above.

- **Java servlet technology:** This is an optimal platform to satisfy the above principles. Not only does it have conceptual advantages over other technologies (CGI, PHP), it also makes XML processing easy because of the good XML suppport in Java.

3 Components

JAPS consists of the following components:

- **A web server:** We use Apache, but other web servers should work as well.

- **A servlet container:** We use Tomcat, but other servlet containers should work as well.

- **A servlet:** We use Cocoon, supplemented by JAPS-specific componenets.

- **A database:** We use PostgreSQL, but other databases should work as well.

The interaction of the components, which is quite standard, is shown in figure **??**.

4 Typical request processing

Let us assume that the browser sends a request for a 'generic' document to the server. The first thing JAPS does is to resolve the 'generic' to a 'real' document ('generic' and 'real' documents are notions from the *theme concept* of JAPS; see above).

Next, JAPS creates an XML representation of the page (we assume a non-binary document). To this end, it retrieves the page content (which is already XML) and metadata from the database and creates an XML document that contains all needed information. This document is then converted into XHTML by means of an XSL transformation and sent to the browser.

To render the page, the browser usually needs a CSS stylesheet. Thus, the browser will send a second request asking for the stylesheet. Now exactly the same three steps as described above will take place in the server: resolving the generic document (CSS stylesheets, too, are generic documents), creating an XML document containing both stylesheet content and metadata (CSS stylesheet contents, too, are stored as XML in the database), and finally the XSL transformation. The only difference is that the last step is a transformation to plain text rather then to XHTML.

The XSL stylesheets are created dynamically too: If the server needs an XSL stylesheet, it sends a request to itself asking for that stylesheet. Then the same steps as above take place with one exception: the XSL stylesheet that transforms the XML to the requested XSL stylesheet is a static one (otherwise, we would obviously run into an infine loop).

With binary documents, only the first step (resolving the generic document to a real one) is performed. After that, the document content is retrieved from the database and sent unchanged to the browser.

Of course it is possible to request a real document directly. In this case, the first step is dropped.

5 Size of Code

- 21700 lines of Java code, including 9900 lines autocoded by XSL transformations,
- 15800 lines of XSL code,
- Data Base contains more than 150 tables.

The line counts are without blank lines and comments.

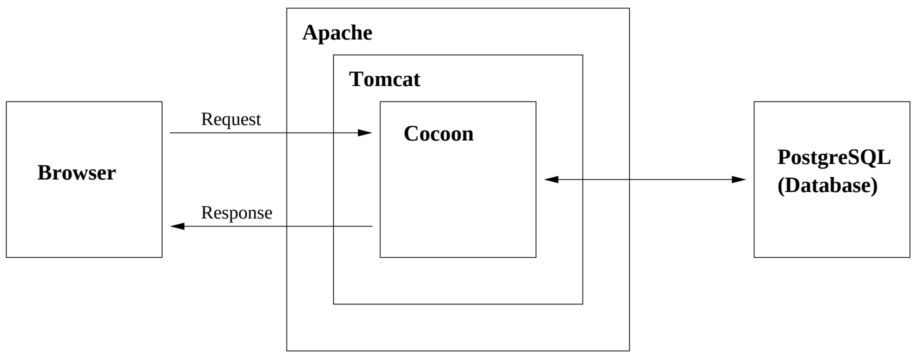

Figure 1: JAPS components

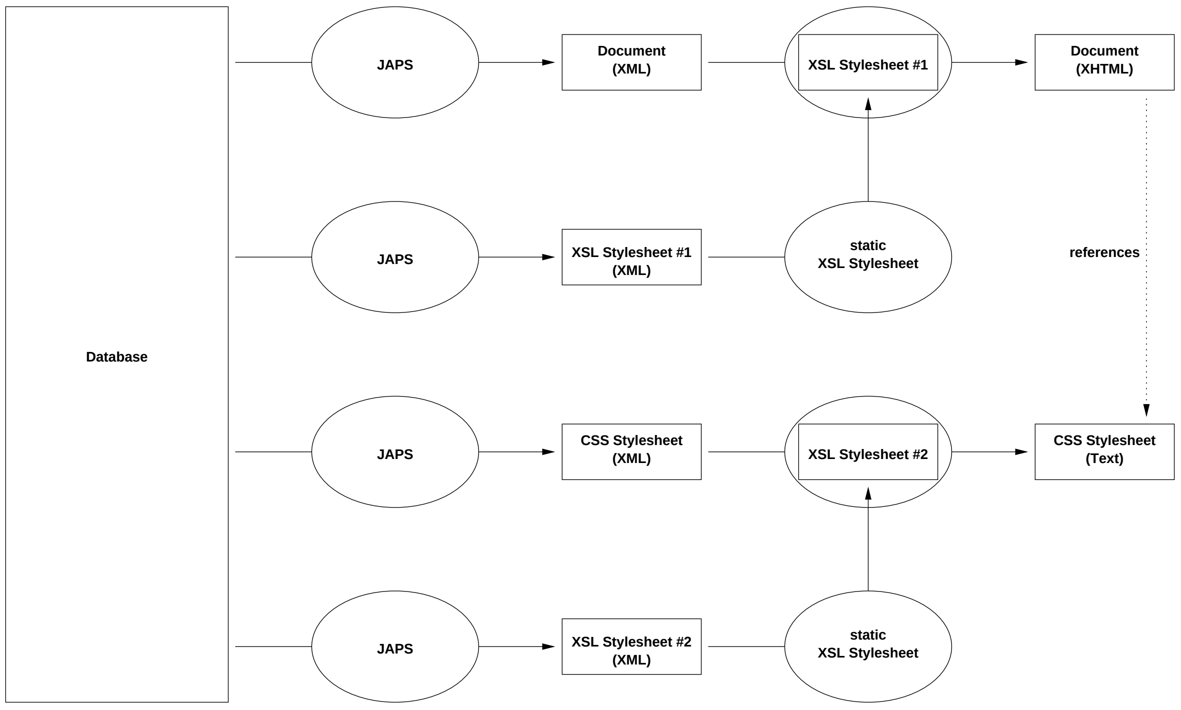

Figure 2: Typical request processing. The resolving of the generic documents is not shown. The picture sketches the dynamical creation of an XHTML document, an associated CSS stylesheet, and two XSL stylesheets needed to convert the XML of the two documents to XHTML and Text/CSS, respectively. The different XSL stylesheets occurring in the picture have the following functions: *XSL Stylesheet #1:* XML-to-XHTML conversion; *XSL Stylesheet #2:* XML-to-Text/CSS conversion; *static XSL Stylesheet:* the fixed (not dynamically created) stylesheet converging XML to XSL.