The Introduction of MUMIE for Supporting Educational Mathematics

Phase 2: Customizing MUMIE for the University of Technology Delft:

Information about the MUMIE project from TU Berlin

Ramonda Golob
Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Delft, the Netherlands

June, 2009

Table of Contents

1 Introduction

This report contains a lot of information regarding MUMIE, an e-learning application. Its specialized applications in mathematical or other courses are numerous. MUMIE consists of several units and a final test. For each unit there are components from which a student can choose. In this report it does not only become clearly that there is a convenient arrangement of the basic components of the content. In the existing running version of TU Berlin, where MUMIE is being developed, there is a distinction between the components Motivation, Application, Definition, Theorem, Algorithm, Lemma, Example and Comment. Not only these elements of the course builder itself, but also a lot of other components are needed in order to keep MUMIE running, which will also be described.

This content has been written in order to make preperations for customizing MUMIE to the set of requirements for educational purposes at the University of Technology Delft. There will also be additional information available at http://www.mumie.net. In the following sections most of the relevant contents of the German documents from a workshop, May 2009, will be elaborated.

Chapter 2 will give general information concerning the MUMIE project. Further on in Chapter 3, the course creator will be explained. Chapter 4 will describe shortly how to create applets. Chapter 5 will explain the use and the knowhow about MMobjects. Further on in Chapter 6, there will be an explanation on how to build and use animations. The use of MUMIE in other languages will be discussed in Chapter 7.

2 MUMIE: General principals

Below, some basic knowledge about the Mumie system is given.

2.1 Documents and pseudo-documents

In Mumie, documents are entities with textual or binary content. (An exception are the so-called generic documents, which have no content. They are explained below.) Examples for documents are theorems, definitions, images or Java applets. Besides ordinary documents, Mumie knows so-called *pseudo-documents*. These are entities which can be treated formally as documents, but are not documents in the actual sense of the term. Examples are: users, user groups, semesters.

A special kind of documents are the so-called *generic documents*. They have no content of their own; rather, they are placeholders for "real" (i.e., non-generic) documents. These placeholders may be implemented by different real documents in different contexts. Generic documents are a means to implement two concepts in Mumie: internationalization and themes. The concepts are explained in the next section.Mumie has a simple build-in version control system for non-generic documents: 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.

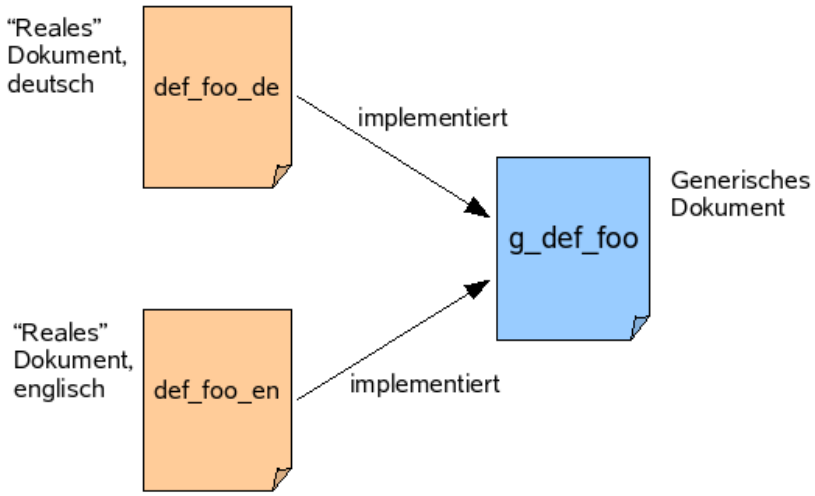

Illustration 1: Generic documents

2.2 Document types

Each document or pseudo-document has a certain type. Types are characterized by names (internally by numbers, too).

Documents with a type name starting with `generic_` are generic documents. All others are non-generic or "real" documents. We also speak of generic and non-generic document types.

For each generic document type a corresponding non-generic one exists. The latter emerges from the former by removing the `generic_` from the beginning of the type name. A generic document can only be implemented by documents of the corresponding non-generic type.

Here follows a list of all document types. First come the non-generic types, then the generic types, both in alphabetical order:

- applet

A Java-Applet. The content is binary; it is a Jar archive containing the applet class file. Sometimes class files are needed by the applet, and supplementary resources like property files. Applets usually have sources which also specify meta-information.

- course

Represents a course as a document rather than a course as an organizational unit (the latter is represented by the pseudo-document class). A course consists of several course sections arranged in a net. The content is XML. Courses are created by the CourseCreater. A GUI tool is used for creating the net in a MUMIE course which is the recommended route for a student, in both logical and chronological way of arranging subjects.

- course_section

A course section, i.e., is a subunit of a course. It consists of elements and subelements arranged in a net, we discussed previously. The content is XML. Course sections are created by the CourseCreater (see course).

- css_stylesheet

A CSS stylesheet. The content is XML (it is translated on-the-fly to the actual CSS format when the stylesheet is requested).

- element

A mathematical content entity. There exist different kinds of elements, distinguished by the category metainfo. Examples are: theorem, definition, motivation. Elements are written in a special TeX dialect which is converted to XML by the mmtex tool. Thus, the actual content is in XML, created from sources in TeX.

- flash

A Flash movie. The content is binary (the *.fla file).

- image

An image. The content is binary (the image data) and may be in any of the following formats: PNG, JPEG, GIF, TIFF (the Mumie administrator can add more when the server is installed).

- jar

A Jar archive. The content is binary (the *.jar file).

- java_class

A single java class. The content is the class file. Documents of this type are used to implement correctors for problems.

- js_lib

A JavaScript library. The content is XML (it is translated on-the-fly to the actual JavaScript code when the library is requested).

- page

An ordinary web page. The content is in XHTML plus an extension XML. Documents of this type are used to implement auxiliary pages like the start page, framesets, etc.

- problem

A mathematical problem. This document type is simular to element. There are three different kinds of problems, distinguished by the category metainfo: applet, mchoice, and traditional. Problems of the first kind are edited in a Java applet. Problems of the second kind contain multiple-choice questions. Problems of the third kind are not edited online; rather, the user submits a written solution to a human corrector. Like elements, problems are written in TeX and translated to XML by the mmtex tool.

- sound

An audio document. The content is binary (the sound data) and may be in the formats MPEG or WAF (the administrator can add more formats at build-time of the Mumie server).

- subelement

A mathematical content sub-entity. Similar to element. Different kinds of subelements exist, distinguished by the category metainfo. Examples are: visualization, remark, example. Subelements are written in TeX and translated to XML by the mmtex tool.

- summary

A summary of a course, course section or worksheet. Written in TeX and translated to XML by the mmtex tool.

- worksheet

A worksheet, i.e., a collection of problems arranged in a net. Three different kinds exist, distinguished by the category meta-information: homework, prelearn, and selftest. The first describes a worksheet given as a homework to the students, the second a prelearning worksheet, and the third a worksheet with which the students can train and test themselves.

- xsl_stylesheet

An XSL stylesheet. The content is in XSLT plus an extension XML.

- generic_css_stylesheet

A generic CSS stylesheet

- generic_element

A generic mathematical content entity

- generic_image

A generic image

- generic_page

A generic web page

- generic_problem

A generic problem

- generic_subelement

A generic mathematical content sub-entity

- generic_summary

A generic summary

- generic_xsl_stylesheet

A generic XSL stylesheet

2.3 Pseudo-document types

Here is a list of all pseudo-documents types in alphabetical order, with a short description for each:

- class

A course in the sense of an organizational unit. A class has lecturers, students attending it, learning groups, and a semester it takes place in (see pseudo-document types *tutorial* and *semester*). Does not represent the contents of teaching; this is done by the document type *course*.

- language

A natural language.

- section.

Similar to a directory on your computer. Sections are explained in more detail later.

- semester

A semester

- theme

A theme (see section *Internationalisation and Themes*).

- tutorial

A learning group with a set of documents to follow by students of a class.

- user

A user, in this case the student.

- user_group

A user group, a group of students enrolled in the same course.

2.4 Meta-information

Each document or pseudo-document has a set of meta-informations or metainfos for short. Not all kinds of metainfo are defined for all types. For example, width and height are metainfos of images, but do not make sense for elements, so elements do not have these metainfos. The value of a metainfo may be a single item or a list of items.

On the Mumie server, metainfos are stored in several database tables. A representation of metainfos as XML exists, too. As a content developer, one will rarely come in contact with either of the both. However, one must specify metainfos in a special format in TeX or Java sources if one writes documents of the respective types. This will be explained in more detail later.

Here is a list of all metainfos, in alphabetical order:

- Category

Defined for: *element*, *subelement*, *problem*, *worksheet*, *java_class*. Distinguishes various “sub-types” of a document type. For example, an element can be a definition, theorem, lemma, application, motivation, or algorithm. This is specified by the category.

- Components

Defined for all non-generic document types. List of documents needed by this document to operate. For example, if an element contains an image or applet, the latter is one of the element's components. Another example is a .jar file needed by an applet: the .jar file is a component of the applet.

- Contained in

Defined for all (pseudo-)document types. The section the (pseudo-)document resides in.

- Content length

Defined for all non-generic document types. Length of the content of the document, in bytes.

- Content type

Defined for all non-generic document types. The internet media type of the content of the document. E.g., text/xml or image/png.

- Copyright

Defined for all non-generic document types. A copyright note.

- Created

Defined for all (pseudo-)document types. The time when the document or pseudo-document was created.

- Deleted

Defined for all (pseudo-)document types. Boolean flag indicating that the (pseudo-)document is deleted. Note: no file what so ever is being erased from the MUMIE database. New and previous versions co-exist in the database.

- Description

Defined for all (pseudo-)document types except *user*. Short description of the (pseudo-)document. Should be suitable to be displayed in a tooltip. If mathematical formulas are inevitable, write them in TeX enclosed in dollar characters.

- First name

Only defined for *user*. The first name of the user.

- ID

Defined for all (pseudo-)document types. The database id of the (pseudo-)document. Unique among (pseudo-)documents of the same type. The value is a non-negative integral number.

- Last modified

Defined for all non-generic document types and all pseudo-document types. Time of the last modification of the document or pseudo-document.

- Links

Defined for all non-generic document types. List of documents linked to this document. Example:

An element contains a hyperlink to another element. Then the latter is a link of the former.

- Login name

Only defined for *user*. The login name of the user.

- Name

Name of the (pseudo-)document. Should be suitable to be displayed in a tooltip. If mathematical formulas are inevitable, write them in TeX enclosed in dollar characters. Defined for all (pseudo-) document types except *user*.

- Password

Only defined for *user*. The password of the user. What is actually stored with this metainfo depends on the type of authentication the Mumie server uses. Usually, it is a hash code (e.g., MD5) of the password.

- Pure name

Defined for all (pseudo-)document types. One can think of this as a filename without directory part and suffixes. Under this name, the document or pseudo-document occurs in the database browser. The pure name is also the base for creating filenames when the (pseudo-)document is represented locally on your computer.

- Surname

Only defined for *user*. The surname of the user.

- VC thread

Defined for all non-generic document types. The version control thread the document belongs to. The version control thread comprises all documents which are different versions of each other.

- Version

Defined for all non-generic document types. Each document has its own version number.

2.5 Section structure

Sections are similar to directories on a computer. A section can contain documents and pseudo-documents of any kind, including other sections; conversely, each document or pseudo-document is contained in exactly one section (the only exception is the root section which is explained in a few moments). Cycles with respect to the included-in relation are not allowed. This way, a hierarchical structure of sections emerges. It has a single origin, the so called root section. Thus, the structure has the form of a tree. Each document or pseudo-document can be found at exactly one place in the tree.

There is a certain standard for the section structure. Beyond the standard, the structure can be arranged arbitrarily. Below is an overview of the standard. The sections are specified by their pure names.

```
<ROOT>
|- system
|  |- common
|  |- libraries
|  |- languages
|  |- themes
|  |- document
|  |- pseudodoc
|  |- element
|  |- problem
|  |- course
|  |- admin
|  '- misc
|
|- org
|  |- users
|  |- user_groups
|  |- <institution>
|  |  '- semester
|  |     |- classes
|  |     |- courses
|  |     '- tutorials
|  '- ...
|
'- content
   |- <institution>
   |  '- field
   |     |- media
   |     |  |- applets
   |     |  |
   |     |  '- images
   |     |- problems
   |     |  '- correctors
   |     '- ...
   '- ...
```

The following list gives a short description about what the sections contain:

- system

(Pseudo-)documents which are of a technical nature (e.g., XSL and CSS stylesheets, layout images, themes, languages) or are needed to operate the server (start page, admin page, etc.)

- system/common

Common resources (e.g., XSL and CSS stylesheets needed by several documents)

- system/libraries

Libraries needed by clients (e.g., Jar archives, JavaScript libraries)

- system/languages

Natural languages

- system/themes

Themes

- system/document

Resources for documents, e.g., XSL and CSS stylesheets for rendering documents, for displaying information about documents, or for creating forms to administrate documents. Note that documents of some types have their own subsections in the system section.

- system/pseudodoc

Resources for pseudo-documents, e.g., XSL and CSS stylesheets for rendering pseudo-documents, for displaying information about pseudo-documents, or for creating forms to administrate pseudo-documents

- system/element

Resources for elements and subelements

- system/problem

Resources for problems

- system/course

Resources for courses, course sections, and worksheets

- system/admin

Resources for web frontends to administrate the Mumie server

- system/misc

(Pseudo-)documents which do not fit into one of the above subsections of the systemsection

- org

(Pseudo-)documents of an organizational nature (users, user groups, classes, etc.)

- org/users

Users

- org/user_groups

User groups

- org/<institution>

Everything related to the organizing of teaching in a certain institution. `<institution>` stands for a pure name adapted to the institution, e.g., “tu_berlin” for *Technische Universität Berlin*. If the server is shared by several institutions, there may be one such section for each institution.

- org/<institution>/<semester>

(Pseudo-)documents for a certain semester. <semester> is to be replaced by a suitable notion for the semester, e.g., “susem_2007” for “summer semester 2007”. In particular, this section contains the pseudo-document of type *semester* which represents the semester. There is one such section for each semester.

- org/uni/<semester>/classes

Classes taking place in the semester corresponding to the parent section.

- org/uni/<semester>/courses

Courses for the semester corresponding to the parent section.

- org/uni/<semester>/tutorials

Tutorials taking place in the semester corresponding to the parent section.

- content

The actual mathematical content (elements, subelements, problems, applets, images, etc.)

- content/<institution>

Content made by or maintained by a certain institution. `<institution>` stands for a pure name adapted to the institution, e.g., “tu_berlin” for *Technische Universität Berlin*. If the server is shared by several institutions, there may be one such section for each institution.

- content/<institution>/<field>

A thematic field of the content (e.g., linear algebra, analysis, probability).<field> is to be replaced by the actual pure name of the field (e.g., “linear_algebra”). There is one such section for each field. Each field section is usually the starting point for a deeper section structure which is motivated by field-specific considerations. This field-specific section structure is not subject to the standard. The standard only expects the two subsections “media” and “problems”.

2.6 Input for CVS

There should only be put those files into the CVS, which are needed to generate all data with the mumie tools. For instance, if ons has a text element “Definition of linear independency”, one only needs to check-in the tex source of it, since the associated meta and content files can be generated with this tex source file alone.

2.7 MUMIE packages from CVS

If one wants to checkout packages from the mumie cvs repository on a Linux console, one needs to do the following:

1. Make locally a directory in which one wants to install your mumie package(s), eg.,
2. `> mkdir mumiecvs`
3. Switch to this directory:
4. `> cd mumiecvs`
5. Checkout the package(s) one wants (note: one can checkout several packages at once):
6. `> cvs -d :ext:<yourUsername>@teefix.math.tu-berlin.de:/net/mumie/cvs co -r ver-2-1-0 <package1> <package2> …`
7. For instance, if the user miller wants to get the mathletfactory_lib and the mmcdk2 packages of release 2.1.0 he types:
8. `> cvs -d :ext:miller@teefix.math.tu-berlin.de:/net/mumie/cvs co -r ver-2-1-0 mathletfactory_lib mmcdk2`
9. Additional Notes:
10. If one omits the release option ”`-r ver-<someReleaseNumber>`”, then one will get the package of the head branch, which is usually the unstable code currently under development.
11. In order to build and install a package after this initial checkout, go into the package directory and read the installation instructions in the file named INSTALL or README

3 The course creator

A course consists of several components. From the MUMIE point of view, a pedagogical concept on how to implement MUMIE in the existing course structure has been developed. First there is the preperation for the lecture, the so-called pre-learning phase. This consists of personalized, automatic correction of the given answers during training in MUMIE by a student. Then there is the lecture phase, which could be implemented in Blackboard in combination with the multimedia components from the MUMIE-DB (database). This MUMIE-DB contains the following extensions: XML, MathML, Latex, applets, images, Flash and Audio. The tutorial phase comes next, in which groups are being supported by a tutor as some kind of coach during solving exercises. Then one has a math-laboratory that provides learner centered work, also supported by tutors. The phase homework follows in which a student can broaden his competence, check his achievements and takes advantage of a personalized, automatic correction, demo and training at home. Then one will reach the last phase, called self-study, which will be supported by an interactive script forum.

This is the organisational concept. Now, when a course is being made, technically speaking, the following structure must be present:

-Course	All lectures lined up.
-course_section	One lecture in particular, like “Differential Equations”
-element	A particular element like a definition/lemma
-subelement	example
-worksheet	exercise prelearning
-problem	exercise afterwards

When one starts creating a course, one has to begin at the bottom with the smallest elements, like an applet. Because one has to add it as soon as one creates an example.

Before one can build the course, one would have to check in the particular files from the checkin-tree through the console in the virtual machine.

One does this 'checking in' as follows:

```
content_test/checkin
cvs checkout *bestandsnaam* (na cd packages)
password: 12:12 ua
content-test cd
./build.sh mount-checkin
build sh mmtex mmjava
```

Now one might wonder how one can combine all these things, like corrector/datasheet/mmjava/mmteX/mmobjekte/etc. For example: When the applet sends the filled out answers to the server, a written correcter in Eclipse, a program for writing java code, will find the errors there and sends it back through another new datasheet. This will be explained in paragraph 3.2. An Mmobject makes sure one can attach an event, e.g. mouse, to an object in an applet. For example: in an applet with a vector that has to move by clicking and holding it with a mouse, one will have to use an Mmobject to make sure the coordinates change with the movement of the mouse.

Each and every example refers to a java applet (Java quellen) and a Corrector (Java quellen). An mmteX corrector usually has one argument, which will travel from checkin-tree to the meta information (*.meta.xml). One has to keep in mind that the applet really needs to be visually implemented in an example: the corrector only exists on the server and is therefore not part of the example.

3.1 How to make a new content package

The following steps need to be followed:

1. Create locally a directory package_new for the new package. Name convention is: content_<instition>_<description> e.g.,
2. `mkdir content_tum_elementary_math`
3. Copy from an existing content package the files into package_new
4. buils.sh
5. INSTALL
6. LICENSE

Into this new made directory, one has to:

1. Create the following subdirectories of package_new
2. checkin/content
3. src/java/net/mumie/mathlet
4. src/java/net/mumie/corrector
5. Adapt script build.sh e.g.
6. change name of package: e.g. package_name="content_tum_elementary_math
7. change name of sections: e.g. section_names="elementary_math"
8. Add a file .meta.xml to all directories below content (all directories which carry a non standard name).
    - They contain the information name and description of the directory, e.g.

<mumie:section xmlns:mumie="http://www.mumie.net/xml-namespace/document/metainfo">
<mumie:name>content elementary math
<mumie:description>

content bridge course, based on material by Sepp Dorfmeister
</mumie:description>
</mumie:section>

- Add as new project to the MUMIE-cvs. Note: The .dir files are created by running ./build.sh mount-checkin.

3.2 General approach for creating a course

To build an entire course, one uses a bottom-up approach. One starts with the small pieces and from there on one builds the whole course.

Hence, the principal steps are:

1. Create the content elements, to be seen in the figure below, (definitions, examples, exercises, etc) and transfer them to the server and to the cvs.
2. Create summaries for HA (Haupt Aufgabe) and PL (pre-learning) sheets, course sections and the course, and transfer these to the server and to the cvs.
3. Create the HA and PL sheets and the course sections. Associate content and summaries to them.
4. Create the course and associate summaries, course sections, HA and PL sheets to it.
5. Assign the number of points an exercise is worth. One must also define a time span one will need to finish an exercise.

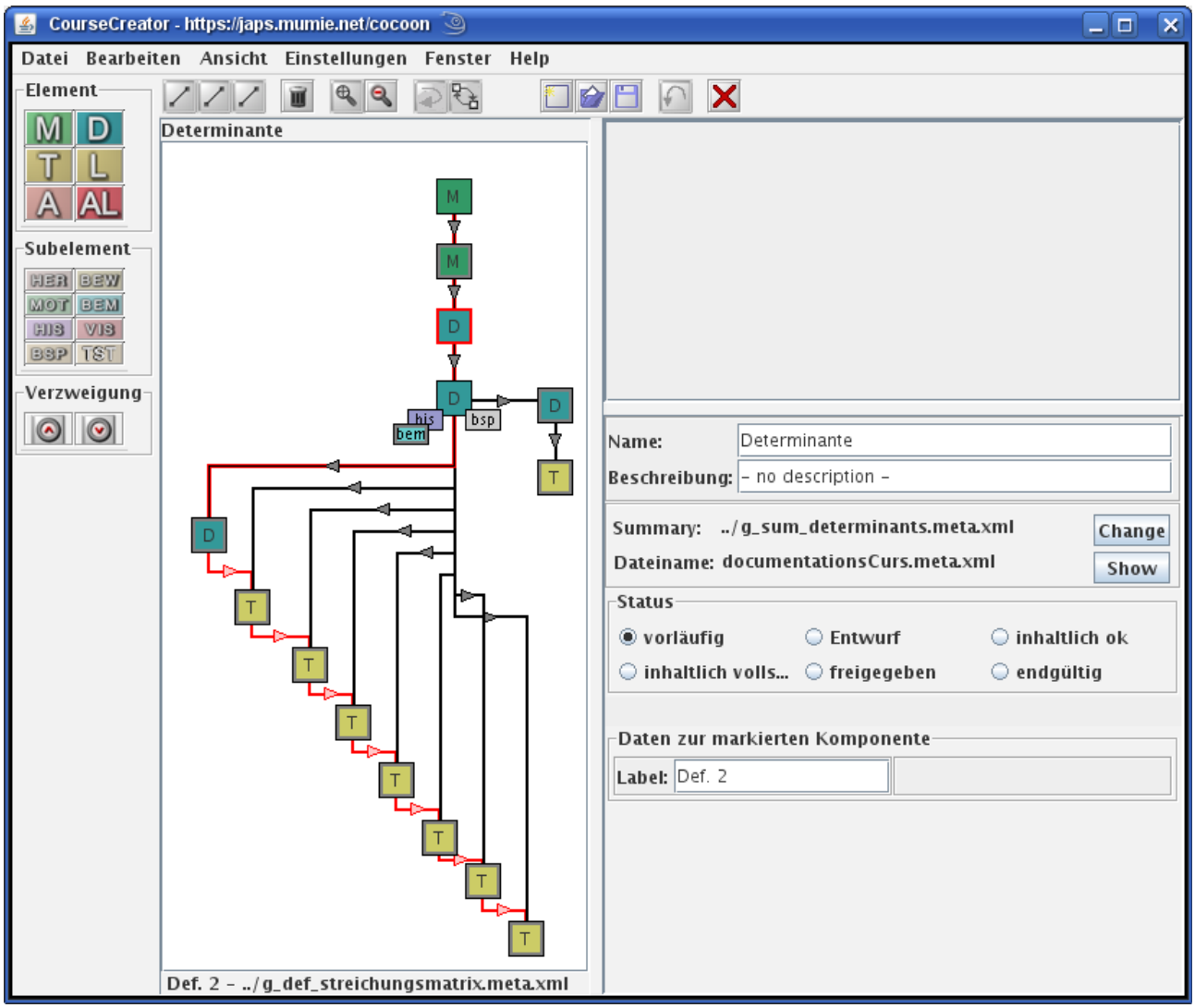

Illustration 2: The course builder with all its elements

All exercises are connected to several datasheets, which will correct the student and will remember the previous answers a student has given. This is to make sure that each student has the same exercises but with different input numbers. Each student has its own personalized datasheet, see the Figure below. This personalized problem data is being saved. This datasheet is connected to an answer datasheet, which will be connected to a correction datasheet. That way there are three levels of datasheets. When the applet starts in the exercise, the datasheets are also being triggered.

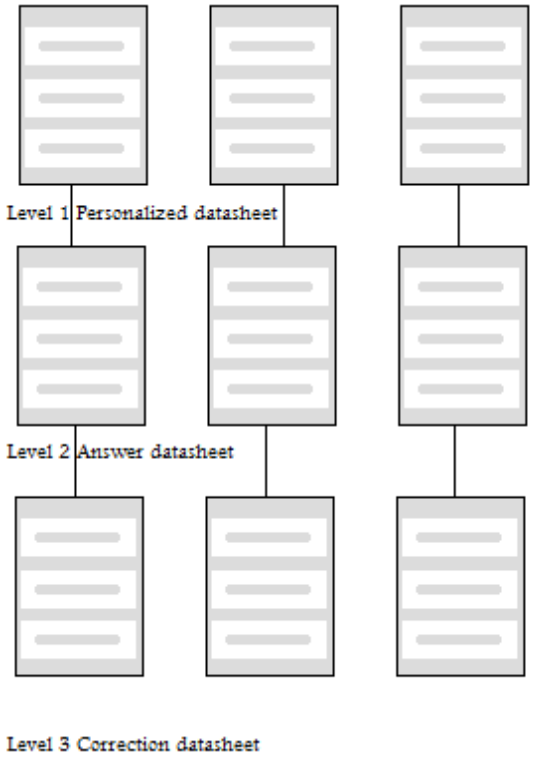

Illustration 3: Datasheets

3.3 Detailled approach for creating text in an example

This information is accesible through the document called, MUMIE-MmTeX, LATEX-Dialekt. This document is only available in a German written printed format, so not yet digital nor English. MmteX quellen are implemented in the Mmcdk, which stands for 'Mumie Content Development Kit'. Here all the new versions are being saved next to the previous ones, instead of the old ones being overwritten.

The most important Mmcdk commands are:
- mmtex “Ubersetzung von TeX-Quellen”
- mmprev “Erzeugen von Previews”
- mmckin “Einchecken auf Mumie-Server”

The global structure of a TeX quelle can be seen in the following lines.

\documentclass[optionen]{japs.typ.kategorie}
\begin{metainfo}
Metainformationen
\end{metainfo}
\begin{content}
Inhalt (Text, Formeln, Bilder, Applets ...)
\end{content}

Then the meta info commands look like this:

\begin{metainfo}
\name{name}
\begin{description}
Beschreibung
\end{description}
\begin{changelog}
Dokumentation der ¨ Anderungen
\end{changelog}
\begin{components}
Deklaration verwendeter Komponenten
(Bilder, Applets, ...)
\end{components}
\end{metainfo}

Then one would also have to bind the applets and images etc. together, by using the following commands. First one would have to declare the components in the meta info part.

\begin{components}
\component{typ}{pfad zur master datei}{binnen id}
Weitere Komponenten
\end{components}

Then one would have to actually insert the components in the content part.
- TeX-Code depends on the type of component
- For images: \image{binnen id}
- For applets: \applet{binnen id} oder applet-
Umgebung
'Binnen ID' is a free to choose reference for the component.

4 Applets

4.1 JAVA applet

Applets
An applet in mumie has several associated files:

1. The java source file **.java**, e.g. `mathletfactory_content/src/java/net/mumie/mathlet/linalg/MatrixTimesMatrixTimesVector.java`
2. The mumie-specific files **.meta.xml** and **.content.jar**, which are located in the ROOT_CHECKIN folder, e.g. (if the ROOT_CHECKIN folder is at `japs2/checkin`)
    - `./japs2/checkin/content/lineare_algebra/media/applets/MatrixTimesMatrixTimesVector.meta.xml`
- `./japs2/checkin/content/lineare_algebra/media/applets/MatrixTimesMatrixTimesVector.content.jar`

1. A copy of the source file **.src.java**, e.g. `./japs2/checkin/content/lineare_algebra/media/applets/MatrixTimesMatrixTimesVector.src.java`

Note that the java source file alone comes from the cvs. All other files are generated during the build process. If we point a tex file to an applet, we need to put the reference to the meta.xml file. We can only check-in a meta.xml file once in a mumie server.

4.2 Mathlet

Mathlets do not contain a multiple language option yet. There are many types of mathlets, called 'Bottom Bereich' and 'Non Canvas Applet'. The applets are being build into the website through an embedding mode, which is directly. Or through a button, so that the applet opens in a new window. You will need to make sure the window has proper width etc.

In an applet that is able to give a demo and a training in which a student can interact, a dynamic reset button is being used. That way the training resets, only the demo remains. When one would want to create a multi language applet, one will have to define keys in order to connect the same words to each other in the preferred languages.

5 MUMIE in different languages

It is very important for a MUMIE community and for the use by foreign universities that the same content is available in different languages. Mumie realizes this by generic documents which are implemented by real documents in different languages.

The *theme concept* allows for different appearances of the same content. A theme represents a certain web layout style. Thus, the theme controls things like fonts, background colors, button shapes, etc. The theme concept is realized by the same mechanism that realizes internationalization, too: generic documents implemented by different real documents for different themes. If the user requests a generic document, the server checks which language and theme the user has selected, and tries to respond with the corresponding real document for the required language and theme.

However, it may happen that the generic document is not implemented for all language-theme combinations. In that case, the server tries to find another real document which is nevertheless suitable to represent the requested generic document. Here are the detailed rules for determining the real document for a given generic document:

1. If a real document exists for the required language and theme, that is selected.
2. Otherwise, if a real document exists for the required language and the default theme, that is selected (see below fore the term “default theme”).
3. Otherwise, if a real document exists for the non-lingual language and the required theme, that is selected (see below for the term “non-lingual language”).
4. Otherwise, if a real document exists for the non-lingual language and the default theme, that is selected.
5. Otherwise, if a real document exists for the default language and the required theme, that is selected (see below for the default language).
6. Otherwise, if a real document exists for the default language and the default theme, that is selected.

The default theme, default language and non-lingual language always exist. The default language can be set by the administrator when the Mumie sever is installed. The non-lingual language represents content which is not language specific. For example, CSS stylesheets usually do not contain any language specific content. The language code for the non-lingual language is “zxx” (language codes are the two- or three-letter codes to identify languages; i.e., “en” for English or “de” for German).

Conclusion

According to research of TU Berlin the math competence among students differs a lot. In order to make MUMIE a success, by which student gain self-assessment of missing pieces of knowledge and competences, students and teachers need to know what the prerequisites for the upcoming courses are. When implementing a test of MUMIE in TU Delft, October 2009, the students need to know what they are able to gain by using it. That way there will be more chance at a good outcome.

Introducing blended learning like this e-learning application, a first step towards automated diagnostic testing is being made. The benefits for the students are: gaining a better understanding of freshman courses and additional testing of common mistakes among multiple choice exercises. More or less only multiple choice answers are asked, because of the correction datasheets which can not cover all sorts of answers. Furthermore, students have access to exercises which have randomized variables, pooled problems, immediate automatic feedback and which can be repeated indefinitely at all times.

TU Berlin has over 200 problems and variants. There has been an evaluation among 130 participants, from which more than 60% said they had used MUMIE in the course. More than 50% said it helped them at better being able to solve problems later on. The perspective of TUB lies in improving feedback, creating more problems and advancing automated diagnostig testing.

It would have been better for TU Delft implementation when multi-lingual dynamically generated webpages would already exist. It would also help if the coursecreator software would have been in English, instead of German. Furthermore, it becomes more clear in this report that in the current structure it is not possible to point out specified tasks for a coursebuilder, preferably the teacher of the course. A course builder would be able to create tex quellen, which will require some guidance for starters. A manual should be made available in time. There has to be a lot of special codes inserted between all kinds of parts, like for example to implement an applet in an example, which are not to be expected from a teacher.

TUB states that MUMIE content is flexible, reusable and sustainable. The authors of MUMIE, a special team, should provide the applets through the MathletFactory into the MUMIE-DB. They should also provide the Latex files, MUMIE-dialect, through a MathMLconverter into the MUMIE-DB. Then the lecturers should be able to edit the semantic nets: this is the part of the course creator which is done by a GUI tool. These are the recommended lines a student can follow in MUMIE by going through a course. These nets are being put into the MUMIE-DB through an XMLconverter. Then all users should be able to access the database browser, from which they will have access to all the multimedia that has been put into the MUMIE-DB.

The goal for the future is to make MUMIE open source, which would be optimal for acedemic environment. An open content database would provide integration of MUMIE content from and

into other projects. It would be flexible and cheap (not for free) to use. In TUB they needed more than 45 Teaching Assistents (60h), in order to provide MUMIE-corrections for a Linear Algebra course.

References

www.mumie.net
documentation/wiki
visited may 2009.

information and documents workshop may 2009,
TU Berlin.