Presentation of Mathematical Knowledge in the ISAC-System

Robert Könighofer
Student at the Institute for Software Technology (IST), Graz University of Technology, Infeldgasse 16b, A-8010 Graz, Austria.
king@sbox.tugraz.at

Bachelor Thesis

Graz, Oct. 2006

Supervisor: Univ.Prof. Dipl.-Ing. Dr.techn. Franz Wotawa
Project Coordinator: Dr.techn. Walther Neuper
Abstract:

ISAC is a system for applied mathematics. This document is a report about the work done in my bachelor project within the ISAC team.

The goal of my project was to design and implement the presentation of the mathematical knowledge to the user. One part of my work comprised design issues which had to be related to the given architecture of the system. This included challenges concerning the highly distributed system, concerning the usage of several rendering devices (standard web-browsers and others) and concerning dynamic generation of HTML.

The other part was about the transformation of XML files into HTML files using the XSLT technology — open for extensions of the language of mathematical formulae to be expected in the future.
Contents

1 Introduction 3
   1.1 About the ISAC-Project 3
   1.2 The Development Phase in Summer 2006 3
   1.3 Description of my Task 4
   1.4 Overview over the Chapters 5

2 User Requirements 6
   2.1 The Different Kinds of Knowledge 6
   2.2 Knowledge Representation 6
   2.3 Links between Knowledge Elements 7

3 Software Requirements 8
   3.1 General Requirements 8
   3.2 The Knowledge Browsers 9

4 Basic Architecture 11
   4.1 Programs and Communication 12
   4.2 The GenHtml Program 12
   4.3 The ISAC-Browsers 12

5 Software Design and Implementation 13
   5.1 Generation of the HTML-Files 13
       5.1.1 Expansion of the KEStore Module 13
       5.1.2 Expansion of the GenHtml Module 15
       5.1.3 Results and Considerations about the Encoding 15
       5.1.4 Testing the stylesheets 20
   5.2 The Knowledge Browsers 21
       5.2.1 Communication between Browsers and Dialogs 24
       5.2.2 Binding a Browser to a Dialog 24
       5.2.3 The Processing of Links 24
       5.2.4 Appearance of the Browser 24
   5.3 Displaying HTML content in a Java GUI 25
       5.3.1 The Processing of Links 26
   5.4 Dynamic Modification of the Static HTML Content 27
Chapter 1

Introduction

1.1 About the ISAC-Project

ISAC is an acronym for “use ISAbelle’s knowledge for Calculations in applied mathematics”. The ISAC-project is a development and research project at the Institute for Software Technology (IST) of the University of Technology in Graz. The goal is to develop a software for applied mathematics, which is supposed to be used for educational purposes. The main feature of the software is, that the mathematical knowledge is transparent to the user. The user is able to see what knowledge is used to solve a problem, and how the knowledge is used. That provides students with the ability to solve problems on their own, while getting help and additional information by the software in an interactive way.

The software consists of an SML part and a Java part. The Eclipse development environment is used as programming environment for the Java part. JUnit is used as test framework (see [HT03] for further information). javax.swing is used for the graphical user interface. A detailed description about javax.swing can be found in [Fis01].

1.2 The Development Phase in Summer 2006

The goal of the developing phase in the summer of 2006 was to make the mathematical knowledge transparent to the user. The active members of the team at this time were Walther Neuper as project coordinator, Alan Krempler as design mentor, and Nebojsa Simic, Georg Kompacher and me as developers.

This developing phase was organized similar to Extreme Programming, which was introduced by Kent Beck in [Bec99]. From the twelve practices, we focused on the following:

1 from http://www.eclipse.org/
2 from http://www.junit.org/
3 from http://java.sun.com/
Continuous Integration and Small Releases: We always had a running system, where new functionality was added piecewise. We implemented a lot of design changes, so it was rather difficult to keep the system running all the time. However, compared to an integration phase at the end of the development, it prevented a lot of problems.

Simple Design: We tried to keep it as simple as possible, but the complex tasks did not always allow simple solutions.

System Metaphor: Metaphors had already been introduced; the challenge was to get familiar with them and to use them strictly.

Whole Team: We were all sitting together while programming. The role of the customer was played by the project coordinator.

Design improvement: We started with an improvement of the whole session management. There was always some mistrust in existing structures.

Coding Standards: Naming conventions were used. The automatic code formatting feature of the Eclipse development environment did also make things much easier.

Test Driven Development: We tried to write JUnit tests before the actual implementation.

To improve the efficiency, we did not make use of the following principles:

Pair Programming: as human resources were too little.

Collective Code Ownership: We did not really plan an ownership for the code. However, after some time, everybody had his special areas of the system.

Planning Game: We only did some design sessions. The planning of the actual implementation was not done explicitly. The coordination could be done while working.

Sustainable Pace: We probably did too much work in too less time.

1.3 Description of my Task

As mentioned in the previous section, the tasks were not separated strictly. However, while working, my main responsibilities turned out to be:

Generating HTML files for the knowledge elements out of XML files.

Finding a possibility to replace some of the HTML elements with dynamically generated information.
• Redesigning the browsers to make these things possible.

These tasks comprised design issues as well as issues of data representation. Different technologies (e.g., RMI, XML-RPC, swing,...) and languages (e.g., Java, XSLT, HTML, XML,...) had to be handled. A lot of requirements, mentioned in the further chapters, had to be met. All changes had to be made with respect to existing structures, which limited the possibilities in the design. On the other hand, a lot of existing components could be reused.

1.4 Overview over the Chapters

This section should give some information about the structure of this document and the content of the following chapters.

2. User Requirements: This chapter explains, which requirements the different kinds of users set. This concerns mainly the representation of knowledge elements as a prerequisite for user interaction.

3. Software Requirements: This chapter makes the requirements explicit, which result from the software components already in use (open source only!), and from the math knowledge fixed by their inherent structure. Most of the listed requirements are about the browsers and their dialogs.

4. Basic Architecture: A brief overview about the architecture of the whole ISAC-system is given. Some components of interest for the understanding of this document are discussed.

5. Software Design and Implementation: This chapter contains a description of the solutions found for the task.

6. Conclusion: The results of the work are summed up. Some developments possible in the future are addressed in short as well.
Chapter 2

User Requirements

There are different kinds of users. Each of them has different requirements on the software. The Userrequirements Document contains all requirements structured by the kinds of users. As only few requirements are relevant to the subtask of displaying knowledge elements, they are structured by topic in the following. Existing requirements belonging to the task were copied from the Userrequirements Document into this document. New requirements were taken into the Userrequirements Document as well.

2.1 The Different Kinds of Knowledge

UR 2.1.1 Each element of the knowledge belongs to either theories, problems or a methods, or to the example collection. The elements belonging to theories are:

- theorems.
- rule sets, i.e. sets of theorems or other rule sets, which are applied as long as they can be applied to a certain formula.
- html data, only containing explanations.

UR 2.1.2 Each element of the knowledge base is displayed in the related browser. This requirement has to be met in particular, if such an element is referenced by a link from another browser: where-ever such a link is located, the element is displayed in the browser it belongs to, a theory-element in the theory-browser etc.

2.2 Knowledge Representation

UR 2.2.1 Examples, theories, problems and methods all have a hierarchical structure and each element of the knowledge base has a unique position in this hierarchy.
UR 2.2.2 *The user can browse through statical HTML-Pages* without e.g. matching a model with a problem.

UR 2.2.3 *An element is always displayed together with the respective location in the hierarchy.* This requirement has to be met in particular, if the element is displayed following a link (independently, from which browser).

UR 2.2.4 *With each element in the knowledge base, explanations can be stored.* Every element in the Knowledge Base can provide explanations illustrating its meaning, giving theoretical background information, referencing related topics or giving examples of use.

### 2.3 Links between Knowledge Elements

**UR 2.3.1 Links can go from any element to any element.** That means, an explanation for a problem can have a link to a method solving it, or to a theorem (in a theory) important for this problem; an explanation for a theorem can have a link to a problem, which uses the theorem in a certain way, etc.

**UR 2.3.2 There are specific links which start an example.** Such a link may be located in any part of the knowledge; if the link is activated, the respective example is displayed in the example-browser (UR 2.1.2) together with the examples location in the example hierarchy (UR 2.2.3) and a worksheet is opened for this example.

**UR 2.3.3 Links outside ISAC’s knowledge base open the standard browser.** i.e. links within the knowledge base may point anywhere, and if the destination is outside ISAC’s knowledge, it is displayed outside ISAC, too.
Chapter 3

Software Requirements

This chapter mainly describes the software requirements for the browsers and the browser dialogs controlling these browsers.

Just like in chapter 2, existing requirements belonging to the task were copied from the Software requirements Document [IT02a] into this document. New requirements were taken into the Software requirements Document as well.

All browsers (Example-Browser and Knowledge-Browsers) present their output in a similar way. Textual descriptions have to be combined with images, formulas, formalizations, problems, ... and links to further informations. All kinds of information might be interlinked among each other.

3.1 General Requirements

SR 3.1.1 **Unique identification by a GUH.** Each item of the examples collection or the knowledge base is uniquely identified by a GUH (Global Unique Identifier). The GUH is a string starting with 'thy_' for theory elements, 'pbl_' for problem elements, 'met_' for method elements and 'exp_' for examples.

SR 3.1.2 **Presentation in a standard browser and in the TSAC-browsers.** The knowledge elements and examples can be viewed in a standard browser as well as in the TSAC-browsers. Thus, knowledge elements have to be available in HTML form in some way.

SR 3.1.3 **GUHs as links.** Links between elements of the knowledge base are defined by the GUH of the link target. To make the links work in a standard browser, some path information has to be added to the URL in the HTML representation of the knowledge elements.

SR 3.1.4 **Asynchronous load of content.** The TSAC-browsers load the content to be displayed asynchronously. That means, that the user interface does not block until the page is loaded. The page is loaded in an extra thread instead.
SR 3.1.5 *The hierarchy is displayed in a frame* in order to have it visible all the time.

SR 3.1.6 *The hierarchy has arbitrary levels.*

SR 3.1.7 *The hierarchy shows the position* of the related element displayed in the browser-window.

### 3.2 The Knowledge Browsers

The following enumerations do *not* show all items contained in the respective SML datastructure; rather it shows the ‘most important’ ones — a preliminary decision, which will be overlayed by filtering due to the dialog-guide.

**SR 3.2.1 A problem page consists of:**

1. the name of the problem or the ’CAS-command’ (a short command similar to an algebra system; e.g. *solve*)
2. a model consisting of the fields ‘given’, ‘where’, ‘find’ and ‘relate’
3. explanations
4. the authors
5. the position within the problem-hierarchy, displayed in the hierarchy-frame

**SR 3.2.2 A method page consists of:**

1. the script
2. a name (only displayed in the hierarchy-frame)
3. a guard consisting of the fields ‘given’, ‘where’, ‘find’ and ‘relate’
4. explanations
5. the authors
6. the position within the method-hierarchy, displayed in the hierarchy-frame

**SR 3.2.3 A theorem page (within theories) consists of:**

1. the name of the theorem
2. the formula of the theorem
3. a link to the proof of the theorem within Isabelle
4. explanations
5. the authors (math authors and course designers)

**SR 3.2.4** *A ruleset page (within theories) consists of:*

1. the identifier of the ruleset
2. the type of the ruleset (Rls, Seq, Rrls)
3. a list of rules and links to the rules. Rules can be theorems, other rulesets or operations.
4. a rewrite order
5. explanations
6. the authors (math authors and course designers)
7. the position within the theory-hierarchy, displayed in the hierarchy-frame

**SR 3.2.5** *A htmldata theory page consists of:*

1. explanations
2. the authors

**SR 3.2.6** *Similar representation for static and dynamic content.* If elements of the knowledge base are shown in the ISAC-browser, the content can be enriched with dynamically generated context dependent information. Any selected formula in the worksheet has a context to an element of the knowledge base. This context information is inserted into the HTML content displayed in the browser if the feature is activated.

**SR 3.2.7** *Data and representation separated.* The knowledge data and its representation should be separated well.

**SR 3.2.8** *Easy generation of different representations.* The system must provide an easy way of generating different representations of the same knowledge data. SR[3.2.7] is a precondition to make this possible.
Chapter 4

Basic Architecture

The current architecture of the ISAC-system is shown in figure 4.1. It has to be taken as starting point of all further considerations. As modifications in this architecture would lead to huge amounts of work, all changes and expansions have to be made with respect to this architecture. Additionally it has to be considered, that the KEStore module will encapsulate a database system in the near future.

![Figure 4.1: Basic Architecture of the ISAC-System](image)

The interesting part for the representation of knowledge elements should be discussed in short.
4.1 Programs and Communication

The Java part is separated into the following programs:

- **The WindowApplication**: This is the front-end where the browsers and the worksheets run in.
- **The ObjectManager**: where the session management, the browserdialogs and the worksheetdialogs run in.
- **The KEStore**: providing access to the examples and knowledge items.
- **The Bridge**: for the communication to the math engine.
- **GenHtml**: for the generation of the HTML files in a batch process.

The communication between the WindowApplication and the ObjectManager as well as the communication between the ObjectManager and the Bridge is done via Java RMI (Remote Method Invocation). This is a Java technology to call methods of objects, which do not have to run on the same Java Virtual Machine, they do not even have to run on the same physical device. Strategies for the RMI usage can be found in [PM01].

Retrieving data from the KEStore is done by use of the XML-RPC technology (see [EGGC00]).

4.2 The GenHtml Program

The examples and the knowledge elements are available in XML files. They can be accessed over the KEStore module, which returns the desired element as an XML string. Martin Lang created a module called GenHtml, which creates HTML files by use of XSLT stylesheets, only working for examples (see [Lan05]). The KEStore and the GenHtml module had to be expanded to be able to process the knowledge elements (theories, methods and problems) as well.

4.3 The TSAC-Browsers

The browsers should be able to show the generated HTML files. However, it is the responsibility of the browser dialog to control the browser. The dialog decides about the content to be displayed in the browser. A page is never loaded directly; it is only loaded if the dialog commands it.

The dynamically generated context to the active formula (to fulfill SR 3.2.6) is coming from the math engine over the bridge to the browser dialog. It is then forwarded to the respective browser if the feature is activated.
Chapter 5

Software Design and Implementation

Solutions to the tasks mentioned in section 1.3, fulfilling the User Requirements and the Software Requirements, are discussed in this chapter.

5.1 Generation of the HTML-Files

5.1.1 Expansion of the KEStore Module

The KEStore module consists of:

- `isac.kestore.KEStore` containing `main()`, which evaluates the passed parameters and starts the KEStore.
- `isac.kestore.KEStoreServices` containing the actual functionality.
- `isac.util.KEStorePath` containing important configuration parameters.
- `isac.util.KEStoreCommunication` which is used for the communication with the KEStore. KEStoreCommunication and the KEStoreServices communicate via XML-RPC.

The examples and the elements of the knowledge base are handled in a very similar way, so the expansion could be done with only a few additional lines of code. It shall not be discussed here in detail. The new arguments to be passed to the `KEStore.main()` method are shown in table 5.1.

Processing of Umlauts: Previously, the KEStore module could not load German umlauts, which occurred in XML files. There were three reasons for that:
Table 5.1: Arguments to be passed to the KEStore.main() method

- The XML files were encoded in iso 8859-15. However, nearly all of the Reader classes in Java expect UTF-8 encoded files. The encoding of the umlauts is different in these two formats. The umlauts have been lost for the first time, in the moment were the file was read. The solution was to use a FileInputReader, which takes the encoding of the file to be read as a parameter.

Listing 5.1: Reading XML Files with iso8859-15 Encoding

```java
private static String readFileAsString(String filePath) throws java.io.IOException {
    StringBuffer fileData = new StringBuffer(1000);
    InputStreamReader reader = new InputStreamReader(new FileInputStream(filePath), "iso8859−15");
    char[] buf = new char[1024];
    int numRead = 0;
    while((numRead=reader.read(buf)) != -1){
        String readData = String.valueOf(buf, 0, numRead);
        fileData.append(readData);
        buf = new char[1024];
    }
    reader.close();
    return fileData.toString();
}
```

Within the iso 8859 family, the encoding of the umlauts is equal, so the above method works for all encodings from this family (iso 8859-1 up to iso 8859-15).

- Umlauts could not be transmitted over XML-RPC from KEStoreServices to the KEStoreCommunication class. This problem is widely discussed
in newsgroups and forums and has to do with the base64 encoding for the transmission. The simple solution was to encode all umlauts with their HTML representation (\&ouml; for the German ö, etc.), and to decode them after transmission. The search for a better solution was omitted because of the lack of time, and the fact, that this module will be replaced by a database system soon.

- Java files did contain the umlauts directly instead of only containing their Unicode number. Different encoding of the Java files (on different platforms) lead to different behavior after compiling. This lesson was paid with a lot of time.

Listing 5.2: The use of Unicode Numbers instead of German Umlauts

```java
// old:
input = input.replaceAll("ö", "\&ouml;" );

// new:
input = input.replaceAll("\u00F6", "\&ouml;" );
```

5.1.2 Expansion of the GenHtml Module

The expansion of the GenHtml program was no big job at all. Existing methods could be reused, so a few additional calls solved the problem. The arguments to be passed to GenHtml had to be expanded. The GenHtml program needs the same parameters as the KESore, so the parameters can be read out of table 5.1.

Additional XSLT stylesheets for the knowledge elements were created. As much information as possible should be shown without a need to scroll, and related information should be shown at related positions. Therefore, HTML tables where used, to structure the content in an according way. No complex XSLT features were used, so the stylesheets can be understood and modified with only a small amount of knowledge about the XSLT technology. All in all, the stylesheets contain about 2 000 lines of code now. Information about the integration of this technology in Java was taken from [Bur01]. For the generation of the stylesheets, [Man02] was used as main reference.

5.1.3 Results and Considerations about the Encoding

This chapter shows some examples for the HTML output of the encoding of the XML files with the created stylesheets. Explanations to the elements and their creation are given. The reader may note, that all the red rectangles and numbers are added for explanation purposes, they are not part of the HTML output.

Method Element: An example output is shown in figure 5.1. The guard, consisting of Given, Where, Find and Relate (2-5) is shown in a table to save space and to make the appearance similar to the CalcHead. Each
field consists of an arbitrary number of MATHML/ISA nodes containing a formula. Each formula is shown in an extra line. The whole table lies within `<div class="context"></div>` tags. These tags are used for replacing the elements between the tags with dynamically generated context information. Details about that are mentioned in section 5.4. Area 6 is used for the content of the EXPLANATIONS node. Some HTML content is copied from the XML file into this section. The author from the AUTHOR tag is copied into section 7 with a leading '(c)'.

The script in 1 needs some extra attention. It is available in the SCRIPT/-MATHML/ISA node of the XML file. This script (and some other XML nodes) contains information, which is separated by line breaks. If the content of the node would be copied into the HTML output one by one, the line breaks would not be visible, viewing the document in a browser. The Hyper Text Markup Language uses `<br/>` for line breaks. Thus, all line breaks in the XML node had to be replaced by `<br/>` on copying the content of the node into the HTML output. An XSLT template was written for this purpose:

```
Listing 5.3: Template for inserting with respect to line breaks
1   <!-- Call the template below to insert the content of a tag with
2     replacement of newlines by `<br/>` as following:
3   
4   
5   
6   
7   
```
This is a recursive algorithm. \texttt{\&#xa;} is the code for the line break in the input file. When the text does not contain any line break, it is inserted directly. If the text contains line breaks, the text before the first line break is inserted first (line 13 to 17). <br/> and a line break is added (line 19). The template is then called with the rest of the text recursively (line 22 to 24). A simple call of this template, as mentioned in the above comment, solves the task of inserting with respect to line breaks.

**Problem Element:** An example output is shown in figure 5.2.

The content of the CAS/MATHML/ISA node is taken as description at ①. The model, consisting of Given, Where, Find and Relate is built in the same way as it is for the methods. The whole table lies within \texttt{\div class=\"context\"} tags again. In Area 6, the HTML formatted content of the EXPLANATIONS node is inserted. The author section ⑦ is equal to the methods again.

**Theoremdata Theory Element:** An example output is shown in figure 5.3.

The theorem name in ① is from the THEOREM/ID node, the formulation of the theorem in ② is taken from the THEOREM/MATHML/ISA node. A table is used for these two elements to make the appearance more appealing. Area ③ contains \texttt{\div class=\"context\"} tags for the dynamically generated context. Section ④ is a link containing the PROOF/EXTREF/TEXT node as text and PROOF/EXTREF/URL as link target. The HTML formatted content of the EXPLANATIONS node is inserted in ⑤. ⑥ is used for the MATHAUTHORS/STRING and COURSEDESIGNS/STRING content if filled. Each of them is inserted with an leading 'c'. The MATHAUTHORS/STRING
gets a following '(math-author)', the COURSEDESIGNS/STRING gets a following '(course-designer)'.

Rulesetdata Theory Element: An example output is shown in figure 5.4. The identifier of the rulesetdata in ① is taken from the RULESET/ID node. ② contains the <div class="context"> </div> tags for the dynamically generated context. The rules are written in a table. The left side contains the content of the RULESET/RULES/RULE/TAG nodes. The right side contains links, where RULESET/RULES/RULE/STRING defines the text and RULESET/RULES/RULE/GUH the link target. ⑥ is taken from the RULESET/ORDER node. ⑦ is the content of RULESET/ERLS/TAG and ⑧ is again a link with RULESET/ERLS/STRING as text and RULESET/ERLS/GUH as link target. ⑨ is the field for the EXPLANATIONS. ⑩ contains finally the MATHAUTHORS/STRING and COURSEDESIGNS/STRING as authors just like mentioned in the theoremdata section.

Whenever a GUH is used as link target, some special processing is necessary. The first three characters determine the kind of knowledge, the referenced
According to SR.3.1.2, the links should work in a standard browser as well. The different kinds of knowledge are located in different directories, named with this first three characters. Additionally, the .html extension has to be added. The GUH `thy_isac_Rational-rls-discard_minus` has to be turned into a link target like `../thy/thy_isac_Rational-rls-discard_minus.html` for example. A template for this processing was written and looks as following:

```
<!--
Call the template below to insert a link. The template expects
that there is a <STRING> tag and a <GUH> tag inside of the
selected element. The content of the <STRING> tag is used as
text of the link, the link is created out of the content
of the <GUH> tag. Call the template as following:
<xsl:call-template name="insert_link">
  <xsl:with-param name="text" select="TAG_TO_SELECT"/>
</xsl:call-template>
-->
<xsl:template name="insert_link">
  <xsl:param name="text" select="."/>
  <xsl:variable name="guh" select="normalize-space(GUH)"/>
  <xsl:element name="a">
    <xsl:attribute name="href">../thy/thy_isac_Rational-rls-discard_minus.html</xsl:attribute>
    <xsl:text>...</xsl:text>
  </xsl:element>
</xsl:template>
```

Figure 5.4: HTML Output of the Encoding of a Rulesetdata Theory Element
The '../' for one directory level up is inserted in line 16. The part of the GUH before the first occurrence of '_' is added in line 17 determining the directory. After a '/', the actual GUH name is inserted followed by the .html extension.

Htmldata Theory Element: An example output is shown in figure 5.5

Figure 5.5: HTML Output of the Encoding of a Htmldata Theory Element

This element of the knowledge base only consists of an EXPLANATIONS node and the information about the authors. These elements were already explained in the previous paragraphs.

5.1.4 Testing the stylesheets

The JUnit tests for the stylesheets encode some special XML files, filled with test data, by use of the generated XSLT stylesheets. The output is checked for correctness.

The check of correctness could be done with a single comparison to an expected HTML output string. This has the disadvantage, that the test may be too strict. If some small changes in an XSLT stylesheet were made, which only
lead to some extra whitespaces in the HTML content and have no effect on the representation in a browser, the test would fail. Checking if the content of the nodes of the XML document appear in the HTML document may be too loose, as the right order of the content is not verified. The solution was to use regular expressions in the following way:

**Listing 5.5: Testing XSLT Encoding with Regular Expressions**

```java
String any = "((\S|\s)+)";
String regex = any+"<html>"+any+"<body>"+any+"<div class="context "+any+
  test_headline">any;
// ...and so on until:
regex += "</html>")+any;
assertTrue("string does not match regular expression", result.matches(regex));
```

This avoids the disadvantage of the two above mentioned methods. The effort of writing such tests is low too. As each line ends with a regular expression matching any string, the search for errors is easy. All lines for the construction of the regular expression can be written as comments first. For one line after the other, the comment prefix is removed. The line which finds the error is that line causing the test to fail, if it is activated.

### 5.2 The Knowledge Browsers

The design is based on the work done in [Gri03] and [Hoc04]. Some new features had to be integrated and some simplifications of the existing design were made. The solution is shown in figure 5.6 and 5.7 and shall be discussed in short. Note, that not all attributes and methods of the classes are shown in the diagrams for simplicity.
The most important change was, to build the browsers totally equal. The way, elements of the knowledge base or the example collection are presented should be equal anyway. The only thing that differs is the behavior of some elements (e.g. the buttons) and their content, they are displaying. The browser itself does not care about the meaning of the elements it is showing. The meaning of the elements are only known by the dialog, controlling the browser. That fact, that the browsers are all built equal can be taken as an indication that the representation of the elements is separated well from their meaning.
5.2.1 Communication between Browsers and Dialogs

The browser and the browser dialog communicate over Java RMI (Remote Method Invocation). Whenever information has to be passed from the browser to the dialog, the notifyUserAction() method of the IBrowserDialog interface is used to pass a UserAction. There are different kinds of UserActions carrying different information. Whenever information has to be passed from the browser dialog to the browser, the IToGuiInterface implemented by the BrowserFrameRMI is used to hand over a UIAction. The UIAction carries a EUIContext, which is implemented as enumeration. It determines, which component of the browser the UIAction concerns. The UIAction is forwarded to this component, where it is finally handled.

5.2.2 Binding a Browser to a Dialog

The binding of a browser to the according browser dialog happens after login. The login() method of the UserManager creates a new Session and returns an interface to this session. The WindowApplication registers to the session by use of the registerBrowserFrame((IToGUI) this) method, passing itself. The session does now execute the initializeWindowApplication() method which calls the openNewBrowserFrame() method for all four kinds of browser, passing the according dialog. A new BrowserFrame is created, taking the according dialog as argument of the constructor. The registerBrowserFrame() method of the dialog is called to register an interface to the BrowserFrame for the dialog.

This explanation might sound frightening at first. However, it leads to a clear separation of the tasks. The browsers do not need to know anything about the dialogs except of an interface to pass UserActions. The dialogs do not need to know anything about the browsers except of an interface to pass UIActions. The WindowApplication does only know the browser but not the dialogs. The session does only know the dialogs but not the browsers themselves.

5.2.3 The Processing of Links

Whenever a link is selected, the Minibrowser creates a UserActionOnLink. How this is actually done, and how the Minibrowser works, can be read in 5.3. The UserAction is sent to the according dialog, where it is evaluated. Whenever a dialog decides to display a link target in its registered browser, an UIActionOnLink containing the link target is sent to the BrowserFrameRMI. The EUIContext of this object is set to UI_CONTEXT_MINIBROWSER, so it is forwarded to the Minibrowser, where the page is finally loaded.

5.2.4 Appearance of the Browser

Figure 5.8 shows the appearance of the browser. The numbers correspond to the elements in figure 5.6 in the following way:
5.3 Displaying HTML content in a Java GUI

Displaying HTML content in a Java GUI is in no big problem since most of the `javax.swing.*` components, which are used to display plain text, are also able to display HTML content. The task of displaying HTML content is done by the

```java
the browser_frame:BrowserFrame
the dynamic buttons (in map_buttons_actions_)
the menu_bar_:JMenuBar
the browser_panel_:BrowserPanel
the mini_browser_:MiniBrowser
the hierarchy_panel_:HierarchyPanel
the horizontal_split-pane:JSplitPane
```
The Minibrowser is shown in figure 5.6. The Minibrowser is an extension of the `javax.swing.JEditorPane`. The `JEditorPane` is per default able to deal with the following content:

- text/plain
- text/html
- text/rtf

The type of content can be selected with the `setContentType()` method. There are two methods for displaying HTML content:

- `setText(String html)` shows the passed HTML string directly. The disadvantage of this method is, that the HTML content has to be loaded into a string manually. URLs pointing to the local file system must work as well as URLs pointing to documents available over the World Wide Web via http. Thus, an extra loading-class would be necessary to get the HTML content independent from the location. Additionally, the whole user interaction should not block while the file is loaded, so the loading-class would have to run in an extra thread, as required in SR 3.1.4. All the problems of thread-synchronization would have to be solved. However, the big advantage would be, that the HTML string could be manipulated directly. The content of HTML tags could be replaced or modified by dynamically generated HTML parts with a simple `String.replace()` command.

- `setPage(String url)` loads the content of the passed URL. This works from the World Wide Web as well as from the local file system. The page can be loaded synchronously or asynchronously. Dynamic changes in the static HTML document are not that easy, but they are possible by use of a so called `EditorKit`. This is explained in section 5.4 in detail.

The use of the `setPage()` method clearly results in less work to be done, so it was declared to be the weapon of choice for the MiniBrowser.

### 5.3.1 The Processing of Links

Links are handled with so called HyperLinkListeners. Whenever a link in the `JEditorPane` is selected, the registered HyperLinkListener is called. In our case, the `BrowserFrame` implements the `HyperlinkListener` interface. It is added as the one and only HyperlinkListener of the Minibrowser. If a link is selected, the `hyperlinkUpdate()` method is called. The `BrowserFrame` creates and sends an `UserActionOnLink`. If a new page has to be displayed, an `UIActionOnLink` containing the link target gets to the Minibrowser. It is loaded by use of the `setPage()` method of the `JEditorPane` class.
5.4 Dynamic Modification of the Static HTML Content

According to SR.3.2.6 the content displayed in the knowledge browsers can be enriched with context dependent information. A context refers to the active formula. The reader may have a look at figure 4.1 while reading the following explanation.

The worksheetdialog is the module which knows the active formula. It queries the math engine over the bridge for a specific context (a context to theories, problems or methods) for this formula. The math engine responds on this query with the XML representation of the desired context. The context is parsed, which leads to a Context object (either ContextTheory, ContextProblem or ContextMethod) carrying the information in attributes. This Context object is passed to the corresponding browserdialog. The browserdialog remembers the context as current context and forwards it to the browser.

Figure 5.9: The Use of an EditorKit to Manipulate the Representation
As mentioned in section 5.3, the HTML content itself is never available in a string to be modified directly. The JEditorPane uses an HTMLEditorKit for displaying HTML content. The only possibility for manipulating the representation is to use an EditorKit. The solution is shown in figure 5.9.

An HTMLFactory is used to create components to be displayed out of HTML elements. Instead of the standard EditorKit, a self made EditorKit can be used. A MinibrowserEditorKit is derived from the HTMLEditorKit. The method getViewFactory() is overridden by a method which returns a self written view factory called MiniBrowserViewFactory. The create()-method is called with all HTML elements of an HTML document. The returned View is displayed. This MiniBrowserViewFactory contains a map between HTML tag strings and IHTMLElementRenderer interfaces. If an HTML tag is not in the map, the create() method returns the standard View. If the tag is in the map, the renderElement() method of the corresponding interface is called to get the View. The class behind the interface can create an own object, which is derived from the ComponentView. The createComponent() method can now implement the representation of the HTML element just as desired. This design is based on the 'Factory Method' design pattern explained in [GHJV00].

This design can be used very flexible. Whenever an HTML tag has to be displayed in a special manner, a new ContextRenderer which implements the IHTMLRenderer interface and a View which creates the Component has to be programmed. The tag to be processed and the according Renderer has to be inserted into the map by use of the addElementPresenter() method of the MiniBrowserViewFactory.

In the current state of the ZS4C-software, this feature is only used to replace everything which is inside a <class="context"></class> tag with the context information from the math engine. The string 'context' is inserted together with a ContextRenderer into the map of the ViewFactory. This class creates a ContextView. The createComponent() method of this class returns a Component displaying the HTML representation of the (dynamic) context instead of the static content between the <class="context"></class> tags. The toHtml() method is called on the current_context_. This method returns an HTML string presenting the attributes of the context object.

5.4.1 Examples for the Results

Methods and Problem Elements

The context dependent information to these two elements of the knowledge base are very similar. An example for a context to the problems is shown in figure 5.10.

The guard or the model, consisting of Given, Where, Find and Relate is replaced. Each of these elements can contain more than one formula and is stored as ModelItemList. The ModelItemList is a Vector of ModelItems. Each ModelItem consists of a formula and an status. If the status of a formula is CORRECT, the formula is shown in green, if it is not, it is shown in red. Only if all formulas of a Given, Where, Find or Relate field are CORRECT, the word
on the left side of the table is written in green. If one or more formulas are not CORRECT, the word is written in red.

Figure 5.10: Example for Context Information in a Problem Browser

**Theory Elements**

An example is shown in figure 5.11. The dynamically generated information of the theory elements is located between the description of the theorem and the proof of the theorem. The elements are organized in tables to provide a clear arranged view of the information. Attributes which are not filled with information are left out (together with their description or title) to save space. At the moment, there are still a few inadequateness in the XML string, delivered
from the math engine. Instead of nothing, a `???.empty` string is returned in some cases, leading to some strange HTML output in some cases. However, these small problems will be solved soon.

Figure 5.11: Example for Context Information in a Theory Browser
Chapter 6

Conclusion

6.1 Results of this Work

A big step towards complete transparency of the mathematical knowledge has been made by solving the tasks as described in this document. Browsing through the knowledge elements can be done with any standard browser. Links between the elements make it easy to gain additional information if desired. The possibility of displaying a context of any formula in the currently processed example to any kind of knowledge is a mighty tool for understanding which information and how information is used to solve a problem.

All design and implementation was done in a way, which makes introduced concepts reusable for solving similar tasks. The system is built very open for further expansions.

6.2 Further Development

6.2.1 Use of a Database System

By now, there are more than 1000 XML files representing the mathematical knowledge of the \( I\tilde{S}A\tilde{C} \)-system. Thus, it is very difficult to administrate the knowledge without any assisting tools. A database system with comfortable possibilities for administration will be created soon, to solve this problem.

This will not lead to big changes in the current system. Whenever knowledge elements are needed, the \( I\tilde{S}A\tilde{C} \)-system accesses them over the KEStore module. If this module is modified, keeping the same interface and delivering the same data from a database system, no other modules will have to be changed.
6.2.2 Improving the Data Representation

As the actual data and their representation are separated well, it will be easy to make the HTML representation of the XML data more appealing by only modifying the stylesheets. A lot of improvements for the learner are possible in this area. However, to solve these problems well, experts in didactics will be necessary. Over all, the formulas in the HTML representations are not formatted that good. Formulas containing fractions for example, are written with a '/' and with the expressions in brackets. The reader is more busy decrypting this syntax than with understanding the meaning of the formula. Further developments on MathML and its integration into browsers have to be watched.

If no better possibility can be found to solve the problem, it would also be possible with an EditorKit. Formulas in HTML files could be written in `<div class="formula"> </div>` tags. A special FormulaRenderer could then make some pretty printing.
Bibliography


[GHJV00] Gamma, Helm, Johnson, and Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software.* Addison-Wesley, Massachusetts, 2000. [28]


