International Journal of Artificial Intelligence in Education (2001), 12, 385-407

ActiveMath: A Generic and Adaptive Web-Based Learning Environment [1]

Erica Melis, Eric Andrès, Jochen Büdenbender, Adrian Frischauf, George Goguadze, Paul Libbrecht, Martin Pollet, Carsten Ullrich
DFKI Saarbrücken, D-66123 Saarbrücken, Germany

Abstract. ActiveMath is a generic web-based learning system that dynamically generates interactive (mathematical) courses adapted to the student's goals, preferences, capabilities, and knowledge. The content is represented in an semantic xml-based format. For each user, the appropriate content is retrieved from a knowledge base and the course is generated individually according to pedagogical rules. Then the course is presented to the user via a standard web-browser. One of the exceptional features of ActiveMath is its integration of stand-alone mathematical service systems. This offers the means for exploratory learning, realistically complex exercises as well as for learning proof methods. The article provides a comprehensive account of the current version of ActiveMath.

INTRODUCTION

Because of the enormous development and increasing availability of the Internet, web-based learning systems become more important for long-distance learning, for completing and complementing the traditional teaching, and for supporting life-long learning. They serve as centrally available systems that allow a user to learn in her own environment and whenever it is appropriate to her.

Hence, several web-based learning systems have been developed such as the commercial Learning Spaces[2] or Blackboard[3]. These systems offer fixed multimedia web pages and facilities for user management and communication but most of them lack support of truly interactive problem solving and real user-adaptivity. Moreover, they use proprietary knowledge representation formats rather than a standardized knowledge representation which is exchangeable between systems. Some user-adaptivity is offered by systems such as ELM-ART (Weber & Brusilovsky, 2001) and Metalink (Murray, Condit, Shen, Piemonte & Khan, 1999).

Tools for mathematical problem solving have been developed before, e.g., the dynamic geometry system CabriGeometre (Balacheff, 1993). The academic PACT tutors (Anderson, Corbett, Koedinger & Pelletier 1995; Corbett, Koedinger, & Anderson 1997] and their descendents, the cognitive tutors from CarnegieLearning, focus on mathematical problem solving at school-level in a fixed curriculum and provide predefined feedback and/or simple dialogs. For instance, Ms.Lindquist[4] web-delivers a course for "algebra word problems" which employs human-like dialogs. However, they do not provide a textbook-like reference or web-based system to explore different parts of a curriculum and have specific knowledge representation formats. In particular, representation and (presentation) functionalities are not separated.

In order to provide problem solving orientation and support but also all the other useful features, in Saarbrücken - at the DFKI and the University of Saarland - we are developing the

[1] The project which led to the results presented in this paper was funded by the German Bundesministerium für Bildung und Forschung. The authors are responsible for the content of this publication.
[2] http://www.ibm.com/mindspan
[3] http://www.blackboard.com
[4] http://www.algebratutor.org/

generic, web-based, adaptive, and interactive learning environment ActiveMath. The first area it is used for is mathematics. In particular, the first on-line course we experimented with in ActiveMath has been the college level `Algebra Interactive' (Cohen, Cuypers & Sterk, 1999).

ActiveMath is realized as a client-server web-architecture that can be accessed using standard web-browsers. ActiveMath provides a generic framework that can be filled with instruction content as well as pedagogical knowledge. Because of its modular design it is easily configurable with respect to components. Because of the separate representation of pedagogical knowledge, the system is configurable with pedagogical strategies and therefore a tool for experimentation on appropriate learning settings.

In a nutshell, notable features of the current version of ActiveMath are user-adapted content selection, sequencing, and presentation, support of active and explorative learning by external tools, use of (mathematical) problem solving methods, and re-usability of the encoded content as well as inter-operability between systems.

This article describes these and other features in more detail. It briefly substantiates why certain goals have been targeted and realized. It describes the architecture and explains the components of the distributed system as well as their integration and communication.

PRINCIPLES OF ACTIVEMATH' DESIGN

To begin with we describe some general principles of the system and briefly discuss them from a pedagogical and from a technical point of view. In particular, we discuss the need for explorative problem solving orientation, personalization, for scrutability, as well as for architectural openness, the separation of knowledge representation and functionality, and the use of ontologies and standards.

Pedagogical Goals

ActiveMath' design aims at supporting truly interactive, exploratory learning and assumes the student to be responsible for her learning to some extent. Therefore, a relative freedom for navigating through a course and for learning choices is given and by default, the user model is scrutable (see Kay00), i.e., inspectable and modifyable.

Adaptivity

Most previous intelligent tutor systems did not rely on an adaptive choice of content. A reason might be that the envisioned use was mostly in schools, where traditionally every student learns the same concepts for the same use. In colleges and universities, however, the same subject is already taught differently for different groups of users and in different contexts, e.g., statistics has to be taught differently for students of mathematics, economy, and medicine. Therefore, the adaptive choice of content to be presented as well as examples and exercises is pivotal.

Moreover, web-based systems can be used in several learning contexts, e.g., long-distance learning, homework, and teacher-assisted learning. Personalization is required in all of them because even within teacher-assisted learning in a computer-free classroom with 30 students and one teacher truly individualized learning cannot be realized. ActiveMath's current version provides adaptive content, adaptive presentation features, and adaptive appearance. Each user can take public and private personalized notes as well.

Exploration and Use of Mathematical Services

During the last decades the mathematics pedagogy community recognized that students learn mathematics more effectively, if the traditional rote learning of formulas and procedures is supplemented with the possibility to explore a broad range of problems and problem situations (Schoenfeld, 1990). In particular, the international comparative study of mathematics teaching, TIMSS (Baumert, 1997), has shown (1) that teaching with an orientation towards active

problem solving yields better learning results in the sense that the acquired knowledge is more readily available and applicable especially in new contexts and (2) that a reflection about the problem solving activities and methods yields a deeper understanding and better performance. Therefore, ActiveMath is designed to offer not only multiple choice questions but also more interactive kinds of exercises.

In these exercises, ActiveMath does not guide the user strictly along a predefined expert solution. It can, however, support the student: The integrated mathematical services can provide feedback on the user's activities. In particular, they can check the correctness of her solution. Moreover, the integrated mathematical services, e.g. Computer Algebra Systems (CAS) or calculators, can support the user by automated problem solving. That is, they can take over certain parts in problem solving and thereby help the user to focus on certain learning tasks and to delegate routine tasks. The use of CAS, function plotters, geometry systems etc. has been demanded since long and has partly been realized with stand-alone tools. Those tools make realistic and complex problems an achievable topic for a lesson and allow the students to train fundamental capabilities and skills such as mathematical modeling for realistic problems.

Conveying Proof Planning Knowledge

We believe that teaching mathematical methods and know-how and know-when has to be introduced into mathematics teaching, apart from the traditional axioms, theorems, and procedures. Indeed, first experiments (Melis, Glasmacher, Ullrich & Gerjets, 2001) suggest that instruction materials based on descriptions of mathematics methods yield a better subsequent problem solving performance than traditional (textbook like) instruction material.

Technical Goals

In order to use the potential power of existing web-based systems they need an open architecture to integrate and connect to new components including student management, assessment, collaboration tools, problem solving tools, and dialog system. ActiveMath has an open client-server architecture whose client can be restricted to a browser. This architecture serves not only the openness but also the *platform independence* which is desirable and important. Moreover, the components of ActiveMath have been deliberately designed in a *modular* way in order to guarantee exchangeability and robustness.

Building quality hypermedia content is a time-consuming process, hence the content should be *reusable* in different contexts. However, most today's interactive textbooks consist of a collection of predefined documents, typically canned pages and multimedia animations. This situation makes a reuse in other contexts and a re-combination of the encoded knowledge impossible and inhibits a radical adaption of course presentation and content to the user's needs. ActiveMath' knowledge representation contributes to re-usability and interoperability. In particular, it is compliant with the emerging knowledge representation and communication standards such as Dublin Core, OpenMath, MathML, LOM[5]. This will ensure a long-term employment of the new technologies in browsers etc. The knowledge representation used by ActiveMath is detailed in section 4. Some of the buzzwords here are meta data, ontological XML for mathematics, and standardized content packaging.

HOW DOES ACTIVEMATH WORK

Before we go into details of the system we want to describe how ActiveMath appears for a user and what the architecture and components are behind this appearance. Let's briefly explain ActiveMath's functionalities with an example.

[5] http://ltsc.ieee.org/wg12/

A User's View

Eva, a student of mathematics, wants to learn everything needed to understand group morphisms. She logs on to ActiveMath. As this is the first time, Eva is using ActiveMath, she has to fill a registration form (see Figure 1). where she specifies personal preferences (e.g., field, colorful/grey presentation, preferred language...).

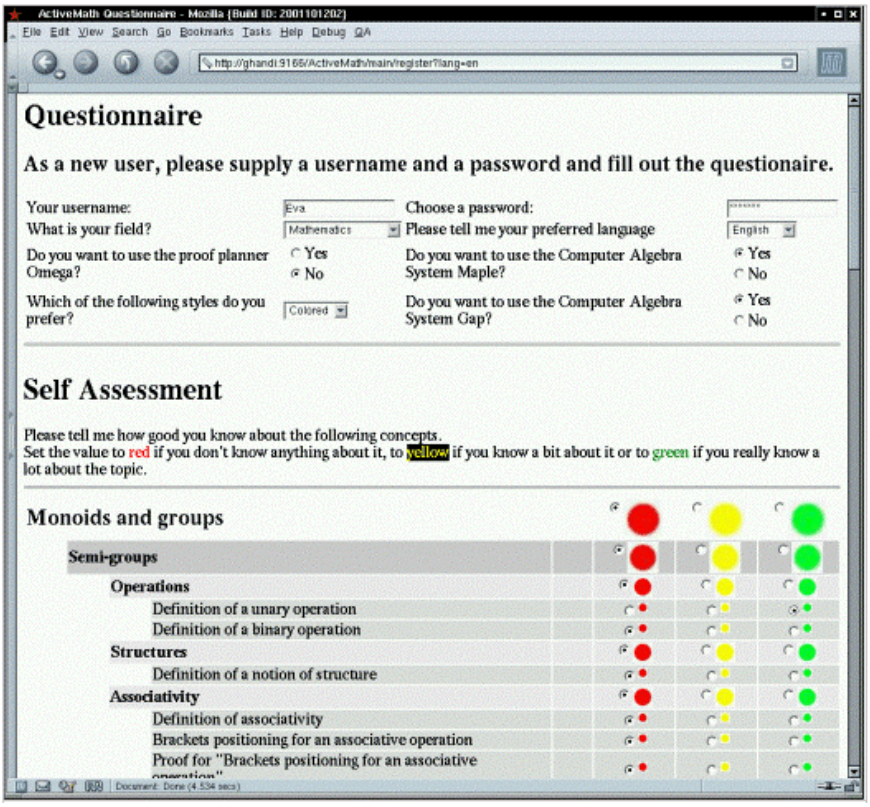

Figure 1. The registration page of ActiveMath

Moreover, she can self-assess her mastery of concepts in the overall course in the hierarchically structured content list shown at the bottom of Figure 1). Then, the main menu is presented to Eva. She chooses her learning scenario and learning goals for that session. (in a school context, a teacher could have chosen scenario and goals). Say, she decides to request a `guided tour' scenario with the goal concept 'morphism'. The required course material (`book') is generated at once. This `book' can be a full course or just the part of a bigger course that teaches today's topic. In another session, she may require another 'book', e.g., the part devoted to the next lesson in school. Eva can also require a predefined 'book'. Each 'book' is a hierarchical structure of pages whose table of content is annotated by colors indicating the user's mastery level. Figure 2 shows a screen shot of a 'book'. On the left hand side of the screen a table of content is displayed. The user can navigate via the linked table of content or next/previous buttons at the bottom of each page.

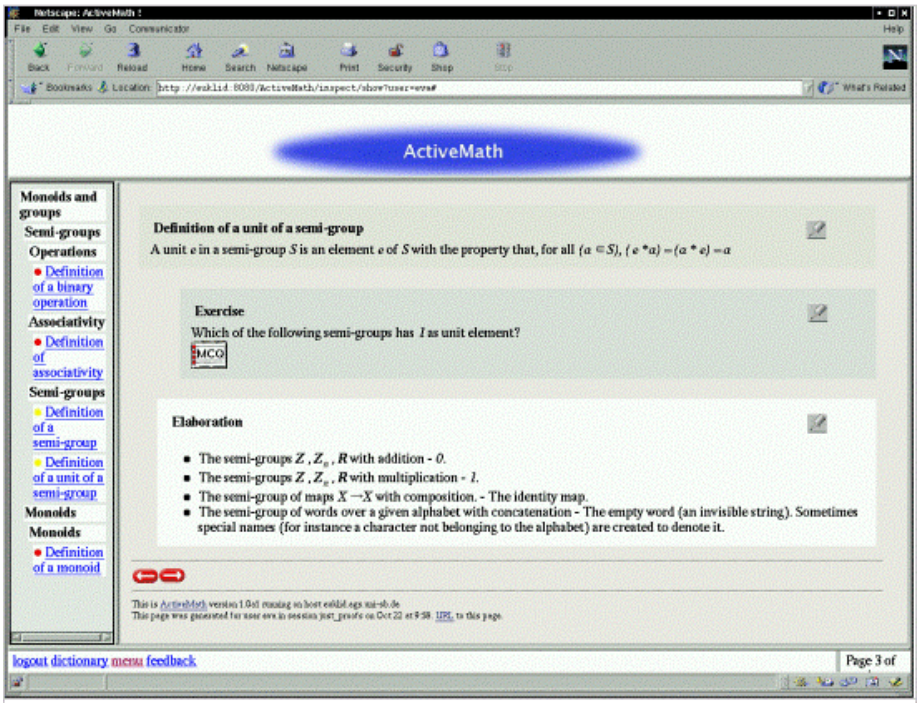

Figure 2: A screen shot of an ActiveMath session

Eva can now browse the book, take notes, look up concepts in a dictionary, work on multiple-choice-questions, and solve exercises by using mathematical systems, e.g., a CAS.

If Eva wants to investigate one of the concepts in the book, she clicks on its occurrence and obtains in a dictionary window (see Figure 3) the course element defining the concept as well as related concepts together with the relations. This way she may learn a larger, more holistic view of the content.

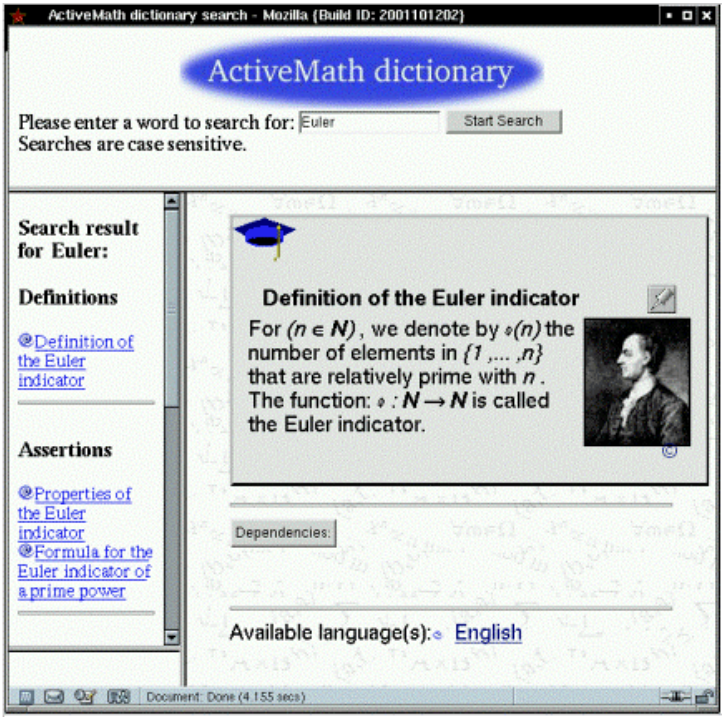

Figure 3. A screen shot of the dictionary

When the user Bert logs on, he may fill the questionnaire differently and may ask for other goal concepts and scenario. If he did not choose the same predefined book as Eva, this user receives another personalized course (see Figure 4) also with other examples and exercises.

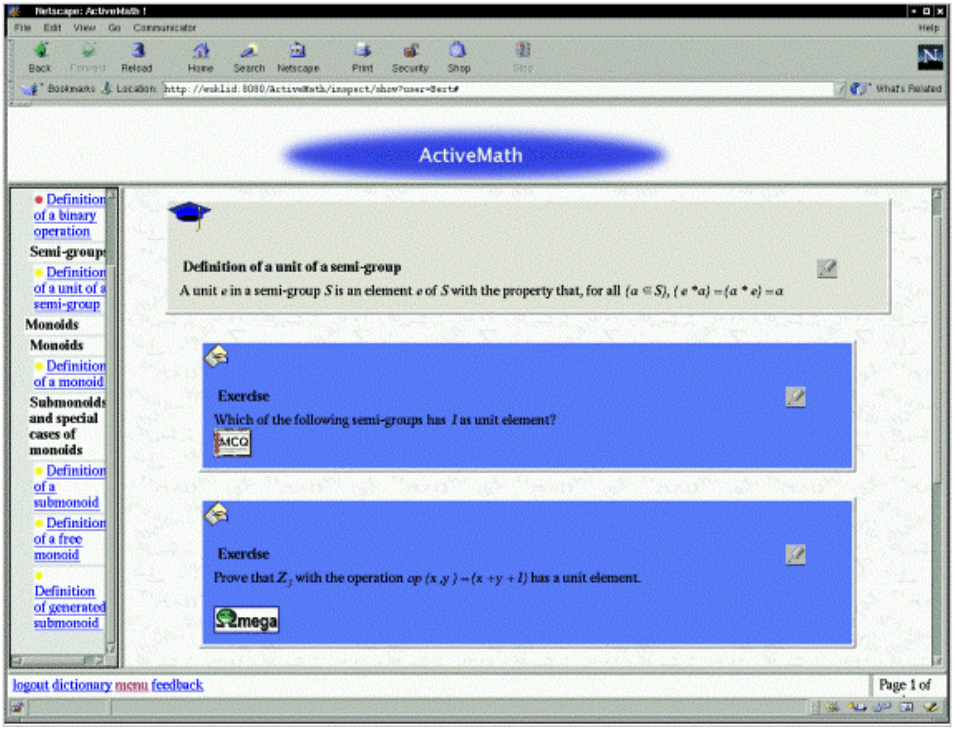

Figure 4. A screen shot of Bert's document

Realization

Now we explain in a bird's eye view how the course generation is realized and which components carry out which task. In what follows, we refer to Figure 5 which gives an overview over the ActiveMath components and their communication.

When a user logs on to ActiveMath, the browser connects to the *web server* which functions as the bridge between the client's browser and the ActiveMath system. The requests from the user and responses from the system pass through it. The web server can fulfill some requests by itself, others are passed to the appropriate ActiveMath components. The web server contacts the *session manager* that sends the questionnaire via the web server to the browser. The information provided via the questionnaire is used to initialize and create a *user model*.

When the user has chosen her goal concepts and scenario, the session manager sends this request to the *course generator*. The course generator is responsible for choosing and arranging the content to be learned. The course generator contacts the *mathematical knowledge base*, MBase (Franke & Kohlhase, 2000), in order to calculate which mathematical concepts are required for understanding the goal concepts, checks the user model in order to find out about the user's prior knowledge and preferences, and uses *pedagogical rules* to select, annotate, and arrange the content - including examples and exercises - in a way that is suitable for the user. The resulting linearized instructional graph, a list of IDs of MBase-items, is sent to the session manager. From MBase the session manager retrieves the actual mathematical content corresponding to the IDs. This content is represented in an xml-format for encoding mathematics. Eventually, the session manager sends the xml-content to a *filter* that transforms the xml-data to -pages which are then presented via the user's browser.

Besides the dynamic generation of a book, ActiveMath offers predefined courses, i.e., courses whose content is predetermined by some author or teacher. Both, the predefined and the dynamically created course are presented to the user as a book which is a hierarchy of pages.

The user's actions are analyzed by (currently very simple) evaluators that calculate updates of the *user model*. When the user logs out, her modified user model is stored.

The *dictionary* facility is based on the semantic information of the knowledge representation. The presentation filter introduces links between a concept in the book and its semantic representation in the knowledge base. From this information, a *dictionary* page is generated dynamically.

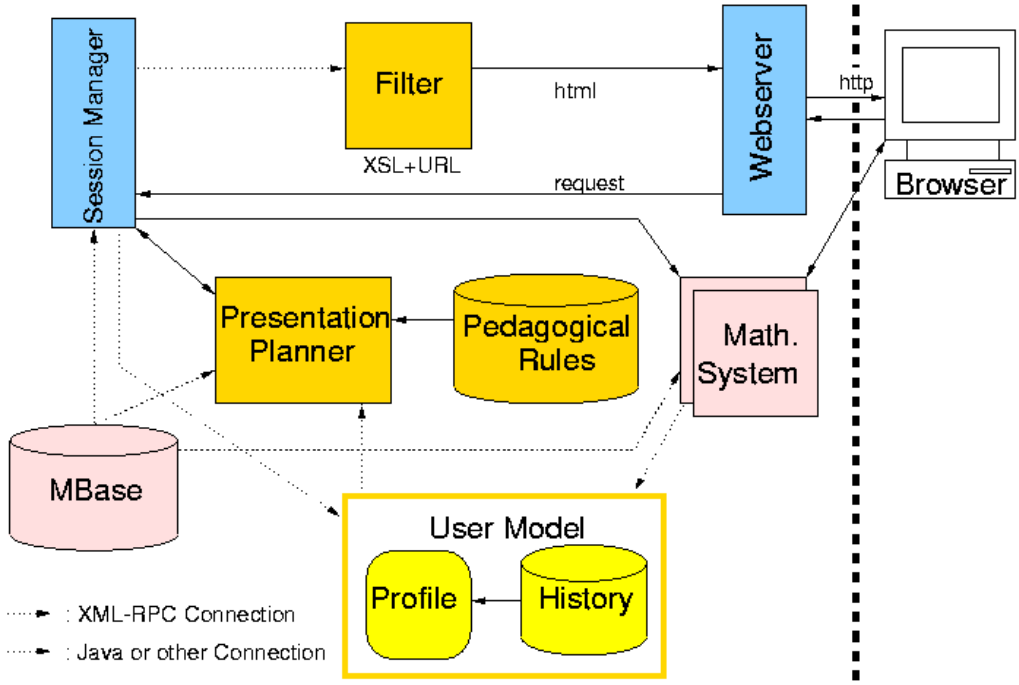

Figure 5. Architecture of ActiveMath

THE KNOWLEDGE REPRESENTATION

As opposed to other interactive textbooks which use a collection of -pages (Cohen, Cuypers & Sterk, 1999) or LaTeX-units (Dahn & Wolters, 2000) ActiveMath uses a *semantic* xml-based representation of mathematical knowledge, an extension of OMDoc (Kohlhase 2001, Kohlhase, 2000).

In ActiveMath the knowledge representation is separated from the system's (presentational) functionalities. This separation is a key for the multiple use of the same knowledge representation in different contexts, for reusing and combining knowledge from different sources, and for managing knowledge with different systems and for different functionalities.

Abstractly, the knowledge representation of the course content is a standardized semantic xml expressing objects and their relations as well as meta data. This is stored in a data base. Moreover, pedagogical knowledge is formalized in so-called pedagogical rules.

In ActiveMath, the content representation consist of items. These objects are either *concepts* or additional items. A course is organized around concepts, i.e., definitions, axioms, assertions (theorems, lemmas, conjectures), proof methods, algorithms. Additional items related to concepts are, e.g., example, exercise, elaboration, motivation, introduction *for* a concept. ActiveMath represents several kinds of relations between concepts (mathematical dependency, pedagogical prerequisites, references) and between concepts and related items such as *for* (example for a concept, exercise for a concept, motivation for a concept, proof for an assertion).

The concepts, related items, and their relations provide a generic mathematical ontology that can be used and *is* actually used by different systems managing mathematical knowledge.

More concretely, our knowledge representation OMDoc is an extension of the OpenMath standard (Caprotti, 1998). OpenMath is a semantic xml-based markup language and a general framework for encoding mathematical *objects*. OpenMath exclusively deals with the representation of the mathematical objects rather than with mathematical documents which may have a complex structure of their own and need additional information such as structures and copyright. Therefore, OpenMath needed several extensions that are realized in OMDoc. Since educational systems need additional pedagogical information we extended the core OMDoc by pedagogical and structural elements such as types of exercises and difficulty of examples and exercises.

OMDoc encodes mathematical objects and facts such as definitions, theorems/assertions, proofs, examples, exercises as well as omtext (remarks, motivations) in an OMDoc document. We refer to all the entities of a document as OMDoc *items*. More specifically, axioms, definitions, theorems/assertions, structures, and proof methods are *concepts*. Every item has an ID. OMDoc items consist of *elements* characterizing this item, e.g., attributes, types, etc.

Figure 6 shows an OMDoc representation of a definition of a monoid. The basic items are symbols, that denote basic mathematical entities. For instance, in the figure the symbol for an ordered-pair, for a unit, etc occur. A symbols points to OpenMath object in a so-called content dictionary (cd), e.g., OMS cd="logic1"..., where logic1 denotes a standard OpenMath content dictionary. The symbols are the simplest items carrying a mathematical meaning. A formula is built from symbols in an OMOBJ element.

Every item can include a natural language formulation as well as formal (OpenMath) objects. In the figure, the CMP (commented mathematical property) element contains the natural language formulation of the definition. It includes OpenMath representations for the objects, e.g., ordered-triple, times, unit, etc. The FMP (formal mathematical property) contains an OpenMath object (a formula), that formalizes the content of the CMP. In the example, the FMP encodes the following logical formula:

$$monoid((M,*,e)) \Leftrightarrow (semigroup(M,*) \wedge unit(e,*))$$

This means, that the triple $(M,*,e)$ is a monoid if and only if the couple $(M,*)$ is a semigroup and e is a unit w.r.t. the operation $*$. Formal objects in OMDoc are necessary to provide a formal input for the external problem solving systems, e.g. for a CAS or for a proof planner.

OMDoc allows the annotation of items and documents with meta data and - for some items - with attributes such as type and for. The meta data scheme of OMDoc is compliant to the DC meta data scheme and includes contributor, Creator, translator, subject, title, description, publisher, date, type, format, identifier, source, language, relation, coverage, and rights. These metadata elements are not mandatory. For the specific needs in ActiveMath, we extended the OMDoc meta data scheme. Currently, the additional metadata elements are depends-on, difficulty, abstractness, and field.

Depends-on contains references to the symbols for all objects that are mentioned in the exercise. Field specifies from which field the content of the item comes. The meta data difficulty and abstractness are still problematic because they have no unique values and thus depend a priori on the author's view. Among others, Figure 6 illustrates the depends-on in the html metadata element that defines dependencies on other concepts. Html depends-on is a relation which is employed by the course generator, as we shall see later.

The relatively verbose, semantic representation buys several clear advantages over - or other purely syntactic representations such as LaTeX, among others,

- it provides an ontology for the content of the course which is indispensable for a reuse of teaching and learning material and for a combination of materials for different purposes.
- It allows for an adaptive presentation which has not to be determined (in advance) in the knowledge representation.

- The presentation formats are pretty flexible rather than restricted to . The OMDoc representation and current technology allow not only for rendering -pages but also other presentation formats such as DVI, SVG, or Flash that can be used, e.g., for high quality printing.
- The meta data annotations needed for learning systems are compliant with standard developments.

```
<definition id="c6s1p4_Th2_def_monoid" for="c6s1p4_monoid" type="simple">
  <metadata>
    <Title xmlns="http://purl.org/DC" xml:lang="en">Definition of a monoid</Title>
    <extradata><depends-on>
        <ref theory="c6s1p1_Th3" name="structure" />
        <ref theory="c6s1p3_Th1" name="semigroup" />
        <ref theory="c6s1p3_Th2" name="unit" />
    </depends-on></extradata>
  </metadata>
  <CMP xml:lang="en" format="omtext">
    A <ref xref="c6s1p1_Th3_def_structure">structure</ref>
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMA>
        <OMS cd="elementary" name="ordered-triple" />
        <OMV name="M" />
        <OMS cd="c6s1p4_Th2" name="times" />
        <OMV name="e" />
      </OMA>
    </OMOBJ> in which
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMA>
        <OMS cd="elementary" name="ordered-pair" />
        <OMV name="M" />
        <OMS cd="c6s1p4_Th2" name="times" />
      </OMA>
    </OMOBJ>
    is a <ref xref="c6s1p3_Th1_def_semigroup">semigroup</ref> with
    <ref xref="c6s1p3_Th2_def_unit">unit</ref>
    <OMOBJ xmlns="http://www.openmath.org/OpenMath"><OMV name="e" /></OMOBJ>
    is called a monoid.
  </CMP>
  <FMP><OMOBJ>
        <OMA><OMS cd="logic1" name="equivalent" />
          <OMA><OMS cd="c6s1p4_Th1" name="monoid" />
            <OMA>
              <OMS cd="elementary" name="ordered-triple" />
              <OMV name="M" />
              <OMS cd="c6s1p4_Th2" name="times" />
              <OMV name="e" />
            </OMA>
          </OMA>
          <OMA><OMS cd="logic1" name="and" />
            <OMA><OMS cd="c6s1p3_Th1" name="semigroup" />
              <OMA>
                <OMS cd="elementary" name="ordered-pair" />
                <OMV name="M" />
                <OMS cd="c6s1p4_Th2" name="times" />
              </OMA>
              <OMA>
                <OMS cd="c6s1p3_Th2" name="unit" />
                <OMV name="e" />
                <OMS cd="c6s1p4_Th2" name="times" />
              </OMA>
          </OMA>
        </OMA>
       </OMOBJ>
  </FMP>
</definition>
```

Figure 6. A definition of a monoid in OMDoc representation

ADAPTIVE PRESENTATION

Every learner is different, and even the same learner will have different goals in different sessions. Therefore, the ActiveMath system offers dynamically constructed courses that suit the learners goals, preferences, and knowledge.

The presentation tools of ActiveMath include a course generator and pedagogical rules employed by the course generator as well as the session manager and presentation filters.

Course Generator

The process of course construction takes different kinds of information into account:

- The goal concepts the user wants to learn. Currently, a learning goal is a concept contained in the knowledge base rather than abstract or pedagogical goals such as improving dependency knowledge.
- The scenario the user chooses. Currently, ActiveMath offers presentation generation for six scenarios: exam, examPreparation, overview, detailedOverview, guidedTour, and detailedGuidedTour.
- The user's knowledge mastery and action history.
- The user's capabilities to work with one of the external systems integrated into ActiveMath.
- Pedagogical rules. The course generator employs pedagogical rules to determine, when which items should be presented and in which order.

The course generator realizes the following steps.

1. Starting from the goal concepts chosen by the user (or by a teacher), all concepts they depend upon are collected recursively. This process uses the depends-on meta data information contained in the OMDoc representation. 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.
2. The second step collects all additional items for the concepts, such as examples. exercises, and elaboration texts.
3. In a third step, pedagogical information represented by pedagogical rules is applied which take information from the user model into account. The rules are used to select and structure the gathered collection of content into an instructional graph. This process is detailed in the next paragraph. For example, in examPreparation definitions, assertions, methods, and exercises will be presented only, whereas in the guidedTour scenario ActiveMath presents examples and motivations in addition.

Moreover, since employing an external system when working on exercises and examples requires a certain minimal familiarity with the systems, ActiveMath presents those exercises only, if the capability is confirmed. In addition, pedagogical information may restrict the available features of an external system. For instance, a student learning about mathematical integration and derivation should not use a CAS to solve his exercises completely, whereas using the CAS as a calculator for auxiliary calculation is acceptable.

4. Finally, the pages are ordered and put into a hierarchy.

The result of the generation is a linearized collection of IDs adapted to the user's needs, preferences, and knowledge that can be transformed to -pages

Pedagogical Rules

The pedagogical rules provide the backbone for the configurability of ActiveMath. Currently, they contain some know-how on which content to present, how to present the content in a user-adapted way, and under which conditions which service systems should be available. So far, this know-how is obvious and simple enough. It is, of course, subject of future research for which we invite other groups to contribute and use ActiveMath as a tool for experimentation .

In the third step of the course generation pedagogical rules are employed to decide

- which additional information should be presented along with a concept,
- which exercises and examples should be presented,
- whether or not to include exercises and examples that make use of a particular external system,
- in which order the information should appear on a page.

For the evaluation of the rules ActiveMath uses Jess (Friedman-Hill, 1997), an expert system shell. In this process, information requested from the user model and the collected IDs of OMDoc items (annotated with the user's knowledge mastery levels) are entered as facts into the JESS knowledge base. Then the rules are evaluated and generate a collection of items to be presented.

The left hand side of a rule specifies the conditions that have to be fulfilled for the rule to fire and the right hand side specifies the actions to be taken when the rule fires. In the following, we provide examples of pedagogical rules for two different types of decisions,[6] where a variable is distinguished by a leading question mark.

Figure 7 shows a rule that fires, if the scenario guidedTour is chosen. In this case, the rules checks whether there exists a fact html(scenario detailedGuidedTour) in the knowledge base of Jess. On the right hand side the action of asserting the items (proofs) ... (exercises) are specified, i.e., these facts are added to JESS' knowledge base which means that the corresponding OMDoc items will be presented, if available. The last action (order motivations … elaborations) specifies the order in which the items will appear on each page.

```
(defrule PatternForDetailedGuidedTour
  (scenario DetailedGuidedTour)
  =>
  (assert (introductions))
  (assert (definitions))
  (assert (elaborations))
  (assert (motivations))
  (assert (assertions))
  (assert (proofs))
  (assert (examples))
  (assert (generals))
  (assert (methods))
  (assert (exercises))
  (assert (order motivations introductions definitions methods assertions proofs
                examples exercises generals elaborations)))
```

Figure 7. A pedagogical rule for determining a pattern for the detailedGuidedTour scenario

Figure 8 shows a pedagogical rule selecting the content for the examPreparation scenario. Compared with a guided tour, only a subset of the available items will be presented.

[6] The rules are a bit simplified for better readability.

```
(defrule PatternForExamPrep
  (scenario ExamPrep)
  =>
  (assert (definitions))
  (assert (methods))
  (assert (assertions))
  (assert (exercises))
  (assert (order definitions assertions methods exercises)))
```

Figure 8. A rule determining the pattern for the examPreparation scenario

Figure 9 shows an example for a rule that chooses exercises with an appropriate difficulty level. If exercises should be presented at all (indicated by (exercises)), and if there exists a definition in the knowledge base of Jess, then d's name is bound to the variable ?definition and the user's knowledge of d is bound to ?user-knowledge. Jess allows to specify Boolean functions (indicated by test) whose value determines whether a rules fires or not. That is, the rule in the figure fires, if the user's knowledge is less than 0.3. In this case, facts are inserted into JESS' knowledge base which in turn trigger the selection of examples for d with difficult levels 0.3, 0.3, 0.5, and 0.7 for the presentation.

```
(defrule RequireAppropriateExercises
  (exercises)
  (definition
   (name ?definition)
   (userKnowledge ?user-knowledge))
  (test (< ?user-knowledge 0.3))
  =>
  (assert (choose-exercise-for ?definition 0.3))
  (assert (choose-exercise-for ?definition 0.3))
  (assert (choose-exercise-for ?definition 0.5))
  (assert (choose-exercise-for ?definition 0.7)))
```

Figure 9. A rule choosing number and difficulty of exercises

The application of the pedagogical rules transforms the heap of IDs that is gathered in the first phase of the course generation to a sorted and grouped selection of material that can be passed to the session manager.

The examples illustrates how nicely the rules can be used to configure ActiveMath. For instance, for a German teaching style the definition and theorems might be presented before the examples, whereas for an American teaching style the examples come first.

Session Management and Presentation

Essentially, the session manager has two main functionalities. The first is the actual realization of the presentation. The second is to store and reload shorthands of documents for resuming a course.

The actual presentation preparation is realized by servlets contained in the session manager, e.g., a servlet calling the course generator for scenario XY. The servlets use URL parameters to react to a web server request and deliver part of a session. These parameters include the name of the user and session in a human readable form.

Currently ActiveMath offers no security and pages of anyone can be read freely even though there is a login mechanism. The simplicity of the request syntax, however, enables a tunneling of them, that means to serve all requests (including exercise proxy-messages, see Section 7) through a single TCP/IP port, thereby allowing access from behind the strongest firewall. This same tunnelling will be used to secure the connections from the client to the server using the secure HTTP protocol.

For every user, each new `book' she opens creates a new session. In order to allow for a return to the same document after a logout, a session state has to be referred to. The information stored about a session has to be sufficient for the return. Therefore, hooked to a session, the following information is stored:

- references to mathematical systems to allow the restart of the exercise in case of a connection failure,
- the linearized instruction graph with page separation to allow the presentation of pages at request,
- names of user and session (for URLs), goals and scenario-name

Content Presentation

Lets go into more detail of the actual presentation preparation which starts with a list of OMDoc IDs. A servlet processes the list, i.e., it fetches OMDoc items from MBase and replaces the IDs by the actual OMDoc content. In particular, the textual content, the CMP, is extracted from MBase and inserted in place of the ID. The resulting big OMDoc is then rendered for the browser through a series of transformations[7] described in the following.

The transformations are produced by several xslt stylesheets[8]: (1) the official stylesheet converting OMDoc to , (2) a stylesheet for transforming mathematical object references to mathematical symbols and (3) the ActiveMath-specific stylesheets for adding features and user-adaptation (for user-adaptive display, appropriate URLs for interactivity, etc.).

The presentation of mathematical symbols is adapted to the user's and possibly to a teacher's preference. For instance the monoid from Figure 6 can be presented symbolically as $monoid(M, *, e)$ but also as $monoid(M, \circ, unit)$.

An ActiveMath-specific transformation enriches the OMDoc representation with dynamic information leading to widget-like features in the final display of the pages. One feature is the adaptive annotation of the table-of-content entries. The color of a bullet indicates the current average mastery level of the entry which is computed from the mastery values of the underlying/included concepts which are retrieved from the user model. Other planned features are a drag-and-drop facility of mathematical expressions (currently symbols only) and a *note* facility that allows to annotate the OMDoc items with private or public notes. Furthermore, depending on the configuration of the presentation, a referencing mechanism can take an OpenMath object and produce a hyper-link from it. Similarly, an explicit ref element can be presented as a hyper-link.

Then xslt stylesheets convert the enhanced OMDoc-pages to . This includes stylesheets used to adapt the -appearance to the user's preferences and to the chosen scenario. Similarly, the presentation of slides needs stylesheets that differ from the stylesheet for a typical book. The last transformation adjusts URLs so that they contain all the session and user information.

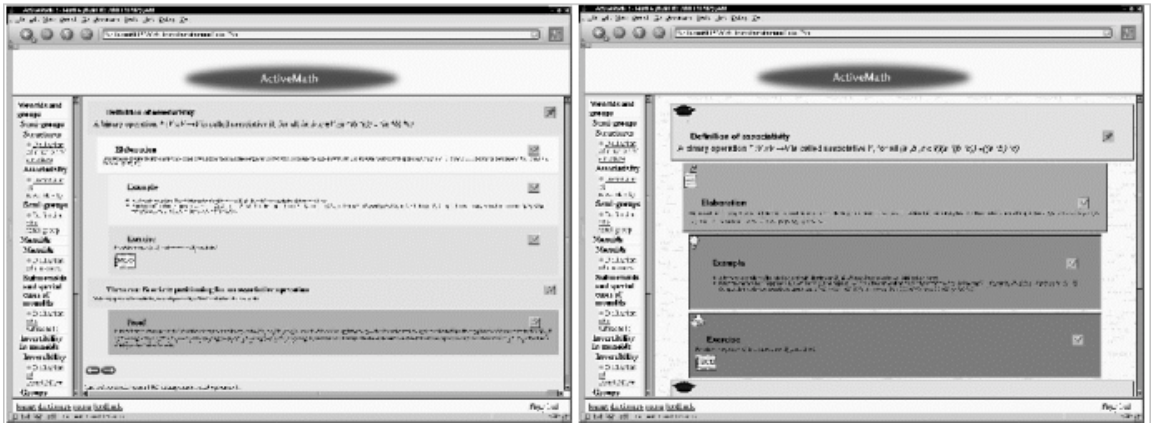

Figure 10. Two different presentations of the same content

[7] These transformations are also called *filters*.
[8] http://www.w3.org/Style/XSL/

Finally, css stylesheets (Bos, Lie, Lilley & Jacobs 1998) determine the general layout, e.g, color, font size, and graphical icons that indicate the type of a paragraph, i.e., indicate whether it is a definition, example, etc. Currently, we have designed two stylesheets, one for a colorful style and the other for a grey style, whose differences can be observed in Figure 10.

Symbol Rendering

The rendering of mathematical symbols and expressions is a general problem for browser delivery in web-based systems. In the past, each mathematical expression had to be presented by a a specific picture. The problem is not totally resolved yet but the OMDoc representation allows for an essential improvement, namely the separation of the presentation from the elements themselves.

Presentation tags for rendering mathematical symbols are contained in OMDoc files. They provide an xslt template that describes, in a parametric fashion, how to produce for the given symbol. The xslt engine is responsible for picking the proper presentation. The browser renders symbols from tags. Currently, we use Unicode for symbol rendering. The rendering and layout is still highly dependent on the browser installation and font availability and thus the resulting presentations differ for different browsers.

Presentation tags can also exist for any other output target. For example LaTeX output is under work together with a serving mechanism of the resulting layout to a vector graphics format. MathML is also considered.

It is important to note that the presentation tags are separated from the elements themselves. Thus it is a relatively easy task to adapt the presentation to another setting or other tastes. Thus ActiveMath achieves the principle of separation of content and presentation.

USER MODELING

As ActiveMath' presentation is user-adaptive, it needs to incorporate persistent information about the user as well as a representation of the user's learning progress. That is, `static' (wrt. the current session) properties such as field, scenario, goal concepts, and preferences as well as the `dynamic' properties such as the knowledge mastery values for concepts and the user's actual behavior, have to be stored in the user model. These different kinds of information are stored separately and hence, the user model consists of the components *history* and static and dynamic *profile*.

The profile is initialized with the user's entries submitted to ActiveMath' registration page which describe the preferences (static), scenario, goals (static for the current session), and self-assessment values for knowledge, comprehension, and application of concepts (dynamic).

How is this information used for adapting the course document? For instance, the `field' property plays an important role for adapting the content and presentation to users who study a topic for the application in different fields. For instance, statistics can be taught differently for mathematicians, computer scientists, biologists, economists, psychologists and certainly with different examples and exercises. Currently, such a diverse statistics course is being prepared. The 'goals' and the 'scenario' essentially determine the items included into the presentation and 'preferences' determines the appearance of the generated document. Currently, the choice of presentation preferences is still pretty simplistic (colorful and grey) but future experiments will provide psychological and instructional evidence for different presentation configurations and appearances that are supportive for different groups of learners.

Currently, the history information is used for the course generation and in a suggestion mechanism. However, this information will as well be valuable for future enhancements of ActiveMath such as dialog and feedback.

User Model Components

Since we did not want to commit once and forever to a particular user model technology but rather be able to experiment with different technologies without having to modify a major part of the system, we specified the abstract interfaces and can thereby integrate different user model technologies and updating functionalities. ActiveMath' distributed architecture supports this implementation strategy. For instance, the history storage can consist of xml-format files (as in a previous version) or of a data base (as in the current version). Similarly, we shall experiment with a table, a data base, and Bayesian Net technology for the concept mastery values.

History

Often, a progress in learning manifests itself not just in tests but in other activities of the user and certainly achievements such as improved collaboration, self-regulation, self-monitoring, and other important skills cannot be judged from the success of exercising only (Fosnot, 1996; Schifter, 1996). Hence, for a more constructivist account of learning it is indispensable to monitor the user's activities.

The history component stores information about the actions the user performed. Its elements, called HistoryAtoms, contain information such as the IDs of the content of a read page or the ID of an exercise, the reading time, the success rate of the exercise. The HistoryAtoms are organized in sessions. Presently, the granularity of a HistoryAtom for reading is the page level because ActiveMath does not yet monitor the user's more detailed reading activities. However, a poor man's eye-tracker is already implemented and this will allow to trace the user's attention and reading time at a more detailed level.

Profile

In order to choose and present the content user-adaptively, the user's preferences have to be stored. And for adapting the content annotations and computing user-adaptive suggestions, information about the user's concept mastery level is required.

To represent the concept mastery assessment, the current (dynamic) profile contains values for a subset of the competences of Bloom's mastery taxonomy (Bloom, 1956):

- Knowledge
- Comprehension
- Application.

```
<Concept>
  <MBaseId>c6s6p3_As2</MBaseId>
  <BloomProperties>
    <Knowledge>0.3</Knowledge>
    <Comprehension>0.1</Comprehension>
    <Application>0.1</Application>
  </BloomProperties>
  <Justification>
    <HistoryStep>5</HistoryStep>
    <HistoryStep>7</HistoryStep>
  </Justification>
</Concept>
```

Figure 11 : An excerpt from the knowledge mastery storage

An entry of the user model is displayed in Figure 11. The user model also stores a justification for each value, that is, a list of pointers to the HistoryAtoms which were responsible for changes of the knowledge mastery assessment. This list indicates, among others, when the changes occurred and why (e.g., whether the changes occurred because the user

modified the mastery value, because she delivered some solution for an exercise, or because she read a text).

The example in Figure 11 represents the mastery of the concept with the ID c6s6p3_As2 which the user knows with a likelihood of 30% and which she can apply with a likelihood of 10%. The justification for this assessment is stored in the HistorySteps 5 and 7.

Updating Mechanisms

Finishing an exercise or going to another page triggers an updating of the user model. Since different types of user actions can reflect and uncover different competencies (of Bloom's classification) they serve as sources for primarily modifying the values of corresponding competencies. In particular, reading concepts corresponds to 'knowledge', following examples corresponds to 'comprehension', and solving exercises corresponds to 'application'. One more competency, transfer, will be tested in the future corresponding to certain exercises.

When the user model receives the notification that a user has finished reading a page, an evaluator fetches the list of its items and their types (concept, example, ...) and delivers an update of the values of those items which depends on the relative reading time. When the user finishes an exercise, an appropriate evaluator delivers an update of the values of the involved concepts that depends on the difficulty and on the rating of how successful the solution was with respect to these concepts.

For user model technologies with a built-in propagation mechanism, the dependencies between the different competencies as well as dependencies of concepts are used to propagate the value changes amongst entries of the user model.

The evaluator itself is an easy-to-exchange component. Currently, ActiveMath can be configured with the following evaluators:

- an incremental updater
- a Bayesian updater (Pearl, 1988).

The incremental updater adds fixed values to the user's mastery assessment. In addition, it applies a function to the incrementation values that takes into account how often the student has already seen this concept. The Bayesian updater increases or decreases mastery values and in addition it propagates among the competency values in a way that simulates a simple Bayesian net. In a Bayesian net user model the propagation will be based on the conditional dependencies of concepts and of competency values.

USE OF EXTERNAL SYSTEMS

Since ActiveMath is designed to support exploratory learning, problem solving tools are integrated. In addition to providing the basic facilities for exploring problems, these systems can be useful because they allow the user to focus on a particular skill to solve a specific problem; they allow the user to explore a problem interactively; in the context of exercises they can provide feedback to the user on where it is promising to explore and where dead ends are reached and they can check the correctness of the user's calculation or derivation and provide diagnose input for the teacher or for an evaluating function. They can check mathematical truth even for a terribly awkward input of the user such as $0.5 + \frac{1}{1+1} = 1$.

Several (mathematical) service systems are available, e.g., CAS such as Maple (Char, Fee, Geddes, Gonnet & Monagan, 1986), Mathematica (Wolfram, 1999), MuPAD (Sorgatz & Hillebrand, 1995), and the freely available GAP (CAS for group theory) (Schönert, 1995) as well as statistics software (SPSS), and calculators. These can be used to efficiently solve computational mathematics problems. Our group has developed another type of mathematical service system, the prototypical proof planner Omega.

The distributed web-architecture of ActiveMath is well-suited for integrating external systems and also the generic, semantic knowledge representation by OMDoc is a basis for

integrating different systems because via translation engines (phrasebooks) these systems can refer to the standard semantic knowledge representation.

In the following, we concentrate on an abstract description of how exercises with external systems are specified and realized in ActiveMath and provide concrete examples for the use of a CAS (Maple) and of the proof planner Omega.

The abstract specification of exercises includes currently:

- the specification of a problem to be solved by the user
- three instructions to the external system:
    - a *startup* containing all the instructions to load libraries, problem data, variable definitions, and other setups,
    - a *shutdown* instruction,
    - *eval* instruction to detect success or failure. *eval* evaluates the user's interaction and returns a success rate (number between 0 and 1) to the proxy and possibly remarks to the user.

What is technically going on when the user chooses to solve an exercise? An user interface is started on the client side. At the server side the external system is started and a *proxy* is started on the servlet server. This proxy is the central server side object for the exercise. It lives in the servlets' virtual machine that is built for each new exercise and for each user. It serves as a bridge between the user interface and the external system. It reports start and end to the user model and it can receive monitoring queries and instructions.

CAS Exercises

Rather than implementing new input editors for every CAS, the current version of ActiveMath offers a classical console, as shown in Figure 12, which requires an input in the language of the given system. This approach relies on the popularity of the CAS and/or on the objective to learn the CAS-language in the course.

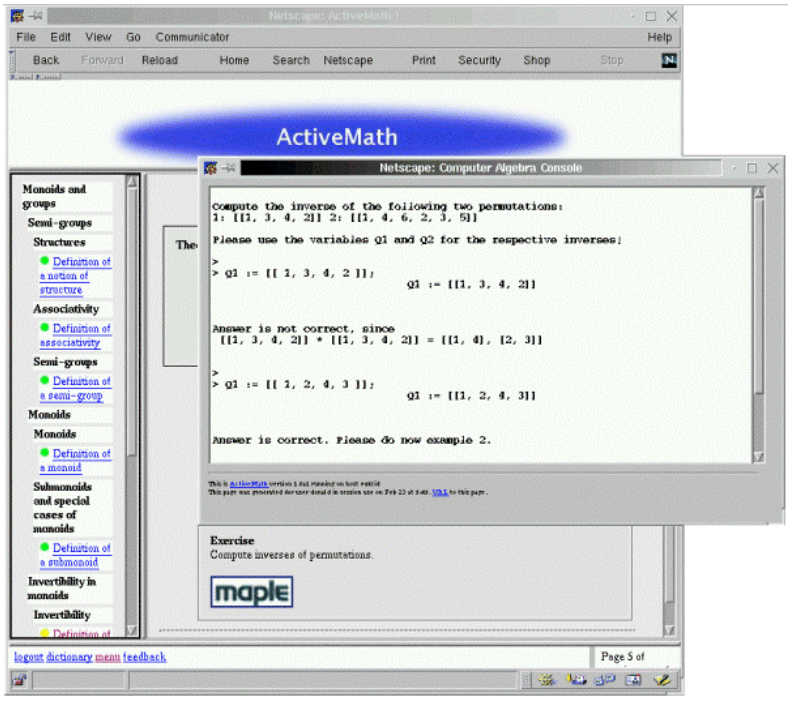

Figure 12: A simple Maple exercise

We have implemented some CAS-exercises with the Maple and MuPAD systems. They ask the user to perform a computation and return the result through a variable whose name is fixed in advance. That is, the problem specified by an exercise is the definition of a variable and the task is to compute and bind the computation result to the variable.

The *eval* instruction is executed after each user input and returns an advise. Since the CAS functionalities could be abused to compute the correct solution, an author can restrict their use in an exercise context.

For a CAS, the abstract features of exercises are specialized as follows.

- The *startup* for a CAS contains the input instructions for the problem, the variable definitions, library loading, user welcoming instructions, and maybe a restriction of the CAS-methods to be used.
- The *shutdown* for a CAS exercise is sent by the proxy to free-up any resource.
- The *eval* for a CAS exercise checks whether the user input is a correct solution, they compute the success rate, determine if the goal is achieved, and may print suggestions or comments to the console.

The proxy of the console applet exercise type for a CAS calculation offers the functionality for a remote teacher to view the exercise activity as well as to provide help messages to the learner.

Proof Planning Exercises

A proof planner is a tool for supporting mathematical theorem proving. It applies so-called methods to perform proofs. These problem solving methods represent typical proof steps such as induction, the application of a theorem, or the simplification of a term. The learning of methods is one of the key issues for Omega's use within the learning environment: a user can learn in which situation which method can be applied, what happens when it is applied, and memorize the methods. Moreover, the automatic expert solver in the background can provide feedback, when the user encounters a dead end in her proof attempt or when an application condition of a method does not hold.

Originally, the proof planner Omega was conceived merely as a proof assistant for logic experts. In an educational environment, however, the typical user has quite different goals and needs, so the tool and in particular its GUI is being modified. A few new features have been introduced into Omega's graphical user interface (GUI). To avoid overloading, we designed an interaction console, shown at the bottom of Figure 13. All the interaction functionality needed to construct a proof is accessible through this window. The user can choose which subgoal to prove next and which method to apply from a list of automatically selected methods. Furthermore variables can be instantiated and proof steps can be backtracked. Whenever the user gets stuck, the automatic proof planner can be called to perform the next step.

The proof in Figure 13 is one of the exercise problems about homomorphisms in the *Algebra Interactive* (Cohen, Cuypers & Sterk, 1999) used as a first testbed for ActiveMath. On the left side the proof tree is shown. The right side can contain either the formal proof, consisting of proof lines with formulae, or its verbalization. In Figure 13 the verbalization of a partial proof is shown too. This multi-modal presentation of a proof may be helpful for students.

The instructions for exercises are concretely specified for a proof planner as follows.

- The *startup* contains the loading instructions for the goal and assumptions, the strategies and methods that are allowed in that exercise, and the theory libraries to be loaded.
- The current, still preliminary, *shutdown* for the proof planner mediates to the proxy the success/failure in terms of proved and open lines and of the time taken for the solution process.
- *eval* instructions for the proof planner are not yet implemented.

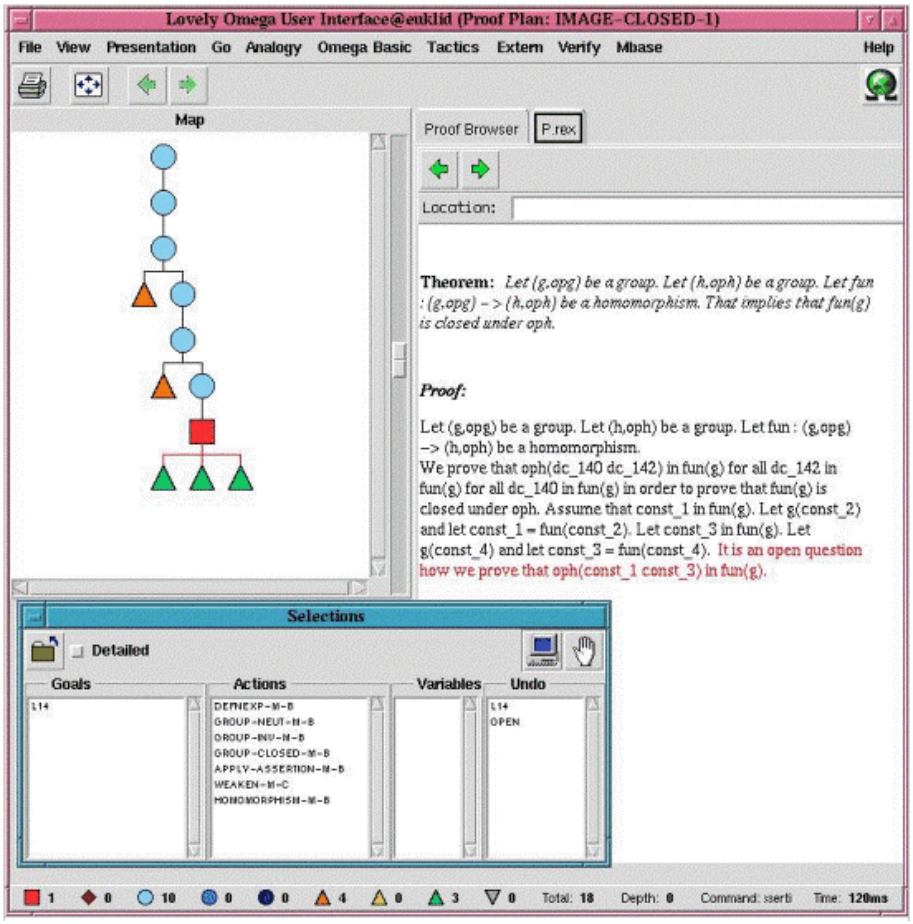

Figure13. The user interface for exercises with Omega.

Communication

In order to use existing mathematical software, ActiveMath needs to communicate with these systems. The central communication component for any exercise activity is a proxy. It is responsible of redirecting messages from the client-side GUI to the wrapper of a computational tool. It informs the user model of the results of the user's activity and allows for monitoring the learner's activity by a teacher.

As a service-management and -addressing infrastructure ActiveMath uses the MathWeb (Franke & Kohlhase, 1999) service broker that directs requests to the appropriate services in a network/Web and provides wrappers for the execution of stand-alone computational tools. The xml-rpc[9] communication with the wrappers made the computational tools available to ActiveMath' servlets.

Uniform Exercise Architecture

The generic exercise architecture is shown in Figure 5. There the arrows indicate the communications, mostly xml-rpc connections. The connections to MBase and to the MathWeb broker are both referred by static URLs, whereas for an exercise the URL of its server proxy is dynamically created.

An exercise can be offered on the client by a Java applet or by any kind of user interface. Its launch is triggered by an http request. This request is handled by the proxy servlet that creates a proxy instance of the proper type and delivers the requested data including parameters

[9] http://www.xmlrpc.org

to connect to the proxy. The proxy triggers the display of the user interface, requests the service of the computational tool needed for the exercise, loads the authoring content from MBase, and sends it to the tool. For more technical details see iamc01.

RELATED WORK

Some related work in the intelligent tutoring systems community is mainly documented in this volume. ELM-ART (Weber & Brusilovsky, 2001) and its descendants are web-based tutoring systems. ELM-ART II was designed for learning programming in LISP and integrates a LISP compiler. Similar to ActiveMath it provides adaptive navigation support by annotating links in a traffic-light metaphor and adaptive sequencing. The courses in ELM-ART are fixed and therefore it is not possible to construct courses according to the user's goals and chosen scenarios as in ActiveMath. ELM-ART has been under development since long and has reached an impressive state of maturity and, as Weber and Brusilovsky point out in this volume, versatility. ActiveMath does not yet offer the same range of communication tools (e.g., discussion lists, chat rooms), but the integration of existing tools is planned.

APHID-2 (Kettel, Thomson & Greer, 2000) extends an existing hypermedia generation system with adaptivity by defining rules that map learner information (e.g., learning style: example) to constraints (e.g. the number of examples per page) that influence the course generation.

The Dynamic Courseware Generator (DCG) (Vassileva, 1997) is similar to the ActiveMath system in that it generates individual courses according to the user's goals and knowledge. In DCG, a planner searches for sub-graphs connecting the goal concept with concepts known by the user. A linearized version of this plan is offered to the user to follow. If a user fails to perform successfully on tests related to a certain concept, a new course plan can be generated. Major differences to ActiveMath are the underlying knowledge representation of the learning material, its usage, and ActiveMath' integration of tools for true interactivity which is not intended for DCG. In DCG, concepts have links to fix -pages that present the actual content to be learned, whereas in ActiveMath the content is *generated* dynamically from the generic representation.

Van Marcke (Marcke, 1998) presents with GTE (Generic Tutoring Environment) an intriguing approach to realise a generic instructional knowledge base. He defines a wide range of instructional goals that correspond to teaching task (e.g., Clarify-Concept) and instructional methods that achieve or decompose the instructional goals (e.g., Clarify-with-Analogy). Currently we are thinking about redesigning our course generator to use a similar planning mechanism (see also vassileva98).

The interactive mathematics textbooks (Cohen, Cuypers & Sterk, 1999; Dahn & Wolters, 2000) use a collection of predefined LaTeX ddocuments or pages and include a fixed set of examples and exercises, the second contains interactive explorations and exercises. In DahnWolter, courses are split to small units (slices) that can be combined in order to construct a textbook. The only adaptivity consists in selecting those pages connected to a certain goal.

Dynamic geometry systems such as CabriGeometre and geolog are neither web-based, nor user-adaptive, and do not generate presentations but they have some other features in common with ActiveMath. In a restricted way, the system geolog (Holland, 1996) employs an approach similar to proof planning in ActiveMath. Holland investigated empirically how students can learn geometrical constructions and proofs in a systematic way similar to what we call (knowledge-based) proof planning (Melis & Siekmann, 1999).

For completeness reasons, we review some relevant commercial systems. It is hard to find commercial solutions comparable to ActiveMath. Two categories of systems have features in common with ActiveMath: (1) Web presentation systems for mathematics and (2) learning environments.

(1) Presenting mathematics on the Web has always been a relatively delicate task. The fine type setting quality required for mathematical documents imposes constraints on previewers of

the presentation. Essentially, such a quality has been achieved by the TeX presentation system which is, however, not targeted to Web presentations and strictly presentational.

Display of mathematics in Web browsers can be made by tools such as Latex2html[10], MathType[11], Maple or Mathematica[12] which provide formulas as pictures or applets. The MathML emerging standard is a better approach but its support by browsers is still insufficient. Still, presentational MathML is a purely presentational approach.

As opposed to presentational approaches, the separation of representation from presentation in ActiveMath employs all the advantages of ontological xml and makes polishing, adaptation for various Web browsers, or the generation of high quality presentations for print (e.g., in Macromedia Flash) a task separated from authoring.

(2) Commercial learning portals such as WebCT[13], Blackboard[14], Learning Spaces[15], or TopClass[16] are mostly designed to provide portals for courses, facilities like chat-rooms, live group-teaching or forums, and administer content and users. As these tools are designed to manage fixed files or their proprietary formats, they cannot adapt their content selection and presentation to the user.

ActiveMath is somewhat complementary to the commercial software in that it presents content on the Web that is represented purely semantically and already offers some of the advantages of such an encoding such as the user-adaptive choice of content. Moreover, it updates the user model, and integrates truly interactive exercises - features that none of these systems offers.

CONCLUSION AND FUTURE WORK

This paper describes the design principles and their realization in the first version of ActiveMath. The system dynamically generates interactive (mathematical) documents according to the user's content needs and presentational preferences and provides facilities for a user interaction with (mathematical) service systems. A demonstration of ActiveMath is available at http://www.activemath.org/demo.

ActiveMath is a generic web-based learning system with a distributed open architecture. It is a shell system that can be filled with content and pedagogical knowledge. It allows to easily configure pedagogical strategies for choosing, presenting, and sequencing learning content, exercises, and examples as well as using scaffolding mechanisms. The open architecture can integrate new components. Currently, a prototypical proof planner and two Computer Algebra Systems are integrated into ActiveMath.

Its standardized xml-representation of knowledge and the dynamic generation of the actual presentation are well-suited not only for interactive mathematics documents but more generally for knowledge-intensive systems whose knowledge acquisition and representation is tedious and should therefore be reused.

Future Work

The research and development of ActiveMath is still in an early stage. Aside from a series of experiments in the near future, we are currently preparing a dynamic suggestion mechanism, a drag-and-drop mechanism that relies on the semantic representation, improved rendering of formulas, the integration of statistics software and multimedia data bases, and an intelligent authoring tool. In the near future, we also plan to integrate some of the existing open-source facilities like chats and user-management into ActiveMath.

[10] http://www.latex2html.org
[11] http://www.mathtype.com
[12] http://www.mathematica.com
[13] http://www.webct.com
[14] http://www.blackboard.com
[15] http://www.ibm.com/mindspan
[16] http://www.wbtsystems.com/

More and diverse courses, multi-modal forms of feedback and an integration of natural language dialogs will follow as well as research on a pedagogically valuable and constructivist use of proof planning.

Acknowledgements

We thank Andreas Franke, Armin Fiedler, Hakim Freihat, Andreas Meier, Volker Sorge, and Jürgen Zimmer for their support in getting ActiveMath off the ground. We are indebted to Michael Kohlhase, the principal developer of OMDoc, for his close cooperation.

References

Anderson, J., Corbett, A., Koedinger, K. & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4(2), 167-207.

Balacheff N, G. M. (1993). Diagnostic et raisonnement explicatif dans cabri-geometre. In B. P. Karsenty L. (ed.), *Explication et cooperation homme-machine : vers la co-construction d'explications* (1-6). CNAM. (http://www.cabri.net/index-e.html)

Baumert, J., Lehmann, R., Lehrke, M., Schmitz, B., Clausen, M., Hosenfeld, I., Köller, & O. Neubrand, J. (1997). *Mathematisch-naturwissenschaftlicher Unterricht im internationalen Vergleich*. Leske und Budrich.

Bloom, B. (ed.). (1956). *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain*. New York, Toronto: Longmans, Green.

Bos, B., Lie, H., Lilley, C. & Jacobs, I. (1998). *Cascading Style Sheets, level 2 CSS2 Specification. W3C Recommendation*.

Caprotti, O., & Cohen, A. M. (1998). *Draft of the open math standard*. (Open Math Consortium, http://www.nag.co.uk/projects/OpenMath/omstd/)

Char, B., Fee, G., Geddes, K., Gonnet, G., & Monagan, M. (1986). A tutorial introduction to MAPLE. *Journal of Symbolic Computation*, 2(2), 179-200.

Cohen, A., Cuypers, H., & Sterk, H. (1999). *Algebra interactive!* Springer-Verlag.

Corbett, A., Koedinger, K., & Anderson, J. (1997). Intelligent tutoring systems. In P. P. V. Helander M. G., Landauer T. K. (eds.), *Handbook of human-computer interaction* (849-874). The Netherlands: Elsevier Science.

Dahn, B., & Wolters, H. (2000). *Analysis individuell*. Springer-Verlag.

Fosnot, C. T. (1996). Constructivism: A psychological theory of learning. In C. T. Fosnot (), *Constructivism: Theory, perspectives, and practice* (8-31). New York, London: Teachers College Press.

Franke, A., & Kohlhase, M. (1999). System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In H. Ganzinger (ed.), *Proceedings of the 16th international conference on automated deduction* (217-221). Springer-Verlag.

Franke, A., & Kohlhase, M. (2000). MBase: Representing mathematical knowledge in a relational data base. In F. Pfenning (ed.), *Proceedings of the 17th international conference on automated deduction*. Springer-Verlag.

Friedman-Hill, E. (1997). *Jess, the java expert system shell* (SAND98-8206). Sandia National Laboratories.

Holland, G. (1996). *Geolog-win*. Dümmler.

Kay, J. (2000). Stereotypes, student models and scrutability. In K. V. G.Gauthier, Claude Frasson (eds.), *Intelligent tutoring systems, 5th international conference* (19-29). Springer-Verlag.

Kettel, L., Thomson, J., & Greer, J. (2000). Generating individualized hypermedia applications. In P. Brusilovski (ed.), *Proceedings of ITS-2000 workshop on adaptive and intelligent web-based education systems* (28-36). Montreal.

Kohlhase, M. (2000). OMDoc: Towards an htmlopenmath representation of mathematical documents (Seki Report SR-00-02). Fachbereich Informatik, Universität des Saarlandes. (http://www.mathweb.org/omdoc)

Kohlhase, M. (2001). OMDoc: Towards an internet standard for mathematical knowledge. In E. R. Lozano (ed.), *Proceedings of artificial intelligence and symbolic computation, AISC'2000*. Springer Verlag.

Libbrecht, P., Melis, E., Pollet, M. Ullrich, C. (2001). Interactive exercises in the ActiveMathlearning environment. In *Issac-2001 workshop on internet accessible mathematical computation*. (http://icm.mcs.kent.edu/research/iamc01proceedings.html)

Marcke, K. van. (1998). GTE: An epistemological approach to instructional modeling. *Instructional Science*, 26, 147-191.

Melis, E., Glasmacher, C., Ullrich, C., & Gerjets, P. (2001). Automated proof planning for instructional design. In *Annual conference of the cognitive science society* (633-638).

Melis, E., & Siekmann, J. (1999). Knowledge-based proof planning. *Artificial Intelligence*, 115(1), 65-105.

Murray, T., Condit, C., Shen, T., Piemonte, J., & Khan, S. (1999). Metalinks - a framework and authoring tool for adaptive hypermedia. In S. Lajoie M. Vivet (eds.), *Proceedings of AIED-99* (744-746). IOS Press.

Pearl, J. (1988). Probabilistic reasoning in intelligent systems: Networks of plausible inference. San Francisco, California: Morgan Kaufmann. (revised second printing)

Schifter, D. (1996). A constructivist perspective on teaching and learning mathematics. In C. T. Fosnot (ed.), *Constructivism: Theory, perspectives, and practice* (73-80). New York, London: Teachers College Press.

Schoenfeld, A. (ed.). (1990). *A source book for college mathematics teaching*. Washington, DC: Mathematical Association of America.

Schönert, M. (1995). *GAP - Groups, Algorithms, and Programming*. Aachen, Germany.

Sorgatz, A., & Hillebrand, R. (1995). MuPAD - Ein Computeralgebra System I. *Linux Magazin* (12/95).

Vassileva, J. (1997). Dynamic course generation on the WWW. In B. d. Boulay R. Mizoguchi (eds.), *Proceedings of AIED'97, 8th world conference on artificial intelligence in education* (498-505). IOS Press, Amsterdam.

Vassileva, J. (1998). DCG+GTE: Dynamic courseware generation with teaching expertise. *Instructional Science*, 26, 317-332.

Weber, G., & Brusilovsky, P. (2001). ELM-ART an adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence and Education*, this volume.

Wolfram, S. (1999). *The Mathematica Book* (fourth edition). Cambridge University Press.