Integration of Mathematical Systems into the ActiveMath Learning Environment

Paul Libbrecht, Adrian Frischauf,
Erica Melis, Martin Pollett, Carsten Ullrich
Saarland University, Germany

Abstract

In Saarbrücken we are developing the *web-based, user-adaptive, interactive learning environment* ActiveMath. Currently, its major features are user-adapted content, sequencing, and presentation, support of active and explorative learning by mathematical systems, support of teachers by information about their students, and a semantic encoding of content that is the basis for reusability.
This article describes how interactive exercises and explorations connected to a mathematical system are inserted into the learning environment and how this simple connection can benefit from other modules of the system.

1 Introduction

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 [14]. In particular, the international comparative study of mathematics teaching, TIMSS [2], has shown 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.

The ActiveMath[1] learning environment [10] is a web-based system. It presents mathematical content in a web-browser. This content is encoded *semantically*, hence is made reusable. The choice of presented material is adapted to the learner's knowledge using pedagogical rules. And, to support explorative learning, ActiveMath integrates interactive exercises, examples, and explorations connected to mathematical systems.

Among other solutions offering web-delivery of mathematical content, ActiveMath appears to be the only system providing a solution for authors to write the content abstractly. The modular architecture loosens the links between the developers and authors. For example, it facilitates the specification of content that an exercise type accepts as input. Compared to most of the systems available, we believe ActiveMath offers an interesting answer to the delicate task of coordinating authors' and developers' works.

[1] A demonstration of ActiveMath is available at `http://www.activemath.org/demo`.

Organization of the Paper

This paper is organized in deepening degree of technicality. We start by presenting an overview of the ActiveMath system together with its user-adaptive capabilities to present content. The general architecture is then refined to focus on the architecture used for the exercises. the technologies used as well as the common exercise conventions are presented. We go on by presenting the two exemplary realizations that have been built to date and conclude with a presentation of current and future works that are made possible by the architecture and content encoding.

2 Overview of the learning environment

2.1 Architecture

Figure 1 depicts the architecture of ActiveMath, i.e., its components and the communications between them (indicated by arrows). It shows the client-server web-architecture with a browser at the client side. Currently, ActiveMath integrates the following components: a session manager, the knowledge base, MBase [7], a presentation planner, a user model, a pedagogical module, and mathematical systems such as the proof planner of ΩMEGA [11] and the Computer Algebra System Maple [5] The systems are connected through their proxies and the MathWeb broker [6].

Requests of the learner and (in the other direction) HTML-pages are communicated via a web-server to the session manager. The session manager stores the generated courses and translates URL requests into actions that are passed to the appropriate component. The presentation planner generates the instructional graph adapted to the learner's goals, preferences, and knowledge by requesting and processing information from MBase, from the user model, and from the pedagogical module. Information about the learner's actions, such as the time intervals of her reading a concept or the success of solved problems, is passed from the session manager or exercise proxies to the user model. The user model updates its values upon receiving this information. We shall describe further details of figure 1 in section 3.

2.2 Knowledge Representation

The knowledge base contains mathematical knowledge represented in the XML-based `OMDoc`[8] format. This allows a fine-grained representation of mathematical content by items such as definition, proof, theorems, motivation, etc. The items may include natural language formulations as well as formal objects. These formal objects (e.g., symbols) relate to actual mathematical objects, i.e., to *semantics*. For instance, independent of whether the presentation is "plus" or "+", both presentations relate to the unique mathematical operation. This semantics provides an ontology for the content of the course which is indispensable for a reuse of learning material and for a combination of material from different sources.

Figure 2 shows an `OMDoc` representation of an exercise invitation. In the figure, the `CMP` (commented mathematical property) element contains the natural language formulation of the definition. The text includes `OpenMath`[4] representations for the objects, e.g., Rn,

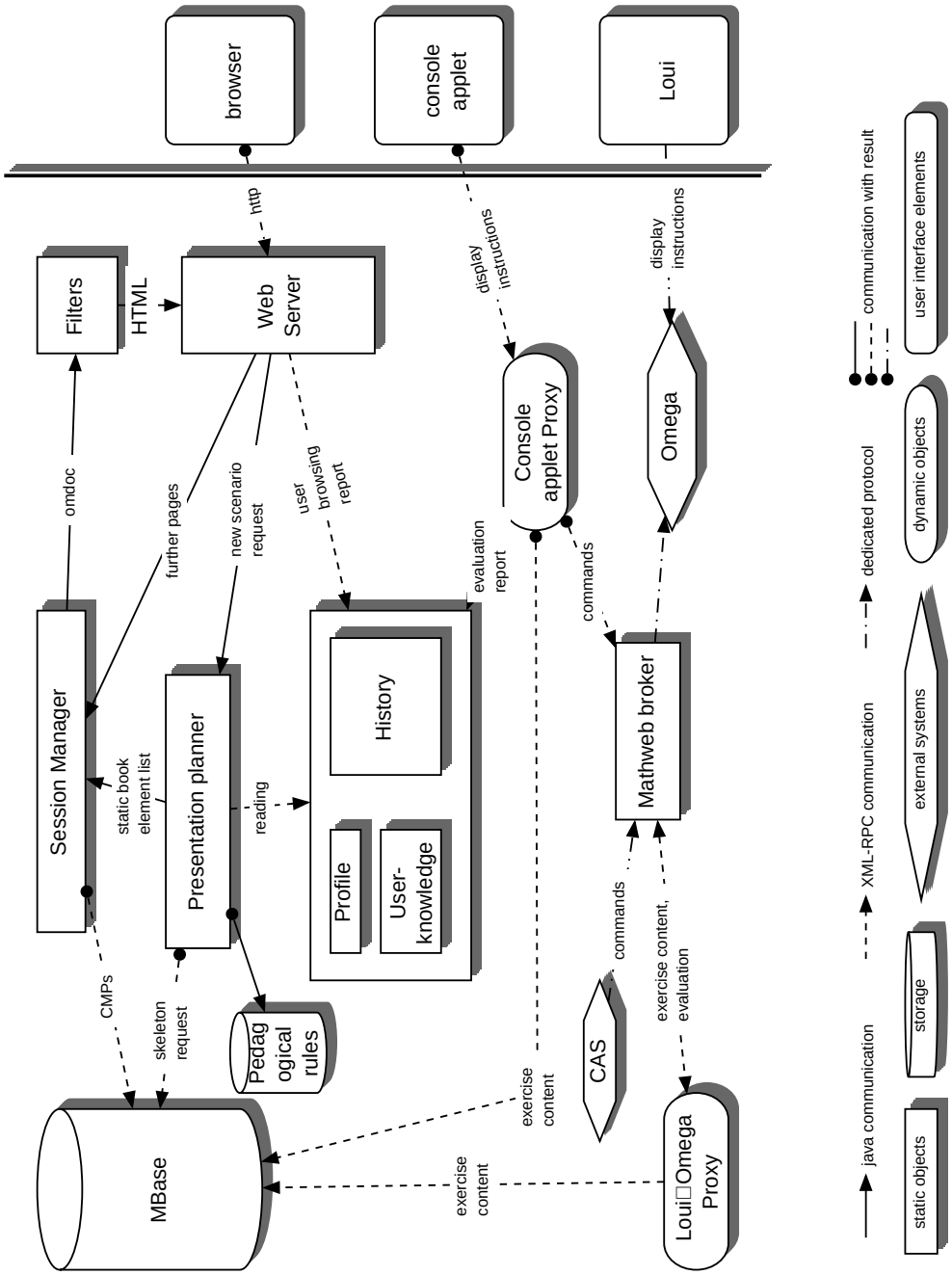

Figure 1: Architecture of ActiveMath

norm, leq, etc. The `OpenMath` content is based on symbols. These are defined to refer to one unique semantic entity. `OMDoc` enables the creation of such symbols in theories. This allows an author to enrich the ontology of his presented course while maintaining consistent semantic throughout the content. Every symbol has its own presentation information which generates XSL stylesheet templates to enable its the rendering. For example, the `OMDoc` from which the example was extracted declares the "boundary of a manifold" symbol, provides presentation (with the ∂) and gives a definition of it.

In addition to the actual mathematical content, our knowledge representation contains meta-data for structures, dependencies, and pedagogical information which can be used for the dynamic generation of interactive documents. The figure displays the `for` attribute of the exercise element that declares which item this exercise is training. This relation is employed by the presentation planner, as we shall see later.

2.3 Presentation Planning

The central component of the ActiveMath content adaptivity is the presentation planner. It generates a personalized course using the meta-data information of the `OMDoc` items and applies pedagogical rules to choose and order the content. The graph of dependencies of the items is used to present items that the learner does not know. The meta-data of exercises, remarks and examples (such as the difficulty and abstractness) is then used by the rules to choose the appropriate items for the introduction of a given concept. The result of presentation planning is presented as a book (called the *static book*) that the learner may freely browse. For more details of the presentation planner, see [9].

2.4 User Modeling

The user model is a component to store, read, and update data about the learner's knowledge. It contains a history of the actions (e.g., the reading of a concept at a certain time), a list of preferences of the learner (e.g., ability to use a certain mathematical system, the choice of a certain stylesheet, or the preferred language), and a list of competence assessments. For each of the concepts in the database, these are represented by values for a subset of the competence features in Bloom's taxonomy [3], namely Knowledge, Comprehension, and Application.

The learner's record of the user model is initialized when first registered. At this time the learner can assert the knowledge for each of the concepts. The user model is, however, updated automatically when the pages are browsed and, most importantly, when an exercise is finished.

3 Exercise Architecture

For a comfortable use of mathematical systems within a Web-based environment ActiveMath realizes a one-click invocation to obtain a user-interface and a back-end system that are loaded and ready to be used. According to traditional web-accessible applications, nothing should be required to be installed on the learner's machine (as we will see, this is not always possible). Also, when designing such systems the breadth of the spectrum of possible

```
<exercise id='ball_boundary' for='bordHn'>
  <metadata>
    <Title>Exercise with the n-dimensional ball</Title>
    <extradata><difficulty level='hard'/></extradata>
  </metadata>
  <CMP>
    Let us call
    <OMOBJ><OMS cd='topDiff_intro' name='ball'/></OMOBJ>
     the set of of points of
    <OMOBJ><OMS cd='topDiff_prerequisites' name='Rn'/></OMOBJ>
     that are at distance maximum
    <OMOBJ><OMI>1</OMI></OMOBJ>
     from the origin:
    <OMOBJ>
      <OMA>
        <OMS cd='relation1' name='eq'/>
        <OMS cd='topDiff_intro' name='ball'/>
        <OMA>
          <OMS cd='set1' name='set'/>
          <OMBIND>
            <OMS cd='set1' name='suchthat'/>
            <OMBVAR><OMV name='x'/></OMBVAR>
            <OMA>
              <OMS cd='logic1' name='and'/>
              <OMA>
                <OMS cd='set1' name='in'/>
                <OMV name='x'/><OMS cd='topDiff_prerequisites' name='Rn'/></OMA>
              <OMA>
                <OMS cd='relation1' name='leq'/>
                <OMA><OMS cd='topDiff_prerequisites' name='norm'/><OMV name='x'/></OMA>
                <OMI>1</OMI>
              </OMA>
            </OMA>
    </OMBIND></OMA></OMA></OMOBJ>

      Prove that       <OMOBJ><OMS cd='topDiff_intro' name='ball'/></OMOBJ>
     is a manifold and that its boundary
    <OMOBJ>
      <OMA>
        <OMS cd='relation1' name='eq'/>
        <OMA>
          <OMS cd='topDiff_intro' name='boundary'/>
          <OMS cd='topDiff_intro' name='ball'/>
        </OMA>
        <OMA>
          <OMS cd='set1' name='set'/>
          <OMBIND>
            <OMS cd='set1' name='suchthat'/>
            <OMBVAR><OMV name='x'/></OMBVAR>
            <OMA>
              <OMS cd='logic1' name='and'/>
              <OMA>
                <OMS cd='set1' name='in'/>
                  <OMV name='x'/>
                  <OMS cd='topDiff_prerequisites' name='Rn'/>
              </OMA>
              <OMA>
                <OMS cd='relation1' name='eq'/>
                <OMA><OMS cd='topDiff_prerequisites' name='norm'/><OMV name='x'/></OMA>
                <OMI>1</OMI>
              </OMA>
            </OMA>
          </OMBIND>
        </OMA>
      </OMA>
    </OMOBJ>
    Do it with <omlet type="loui" function="topDiff_intro_code1"/>.
  </CMP>
</exercise>
```

Figure 2: An OMDoc representation of an exercise in differential geometry which may be presented as: *Let us define* B_n *as the set of points of* $\Re_n$ *that are at distance maximum* 1 *from the origin,* $B_n = \{x | x \in \Re_n \wedge ||x|| \leq 1\}$. *Prove that* B_n *is a manfifold and that its boundary,* $\partial B_n = \{x | x \in \Re_n \wedge ||x|| = 1\}$. *Do it with* ΩMEGA.

exercises an author can develop without programming is an important feature. Apart from being familiar with the mathematical system and its language, the author's task consists in writing data only.

We have developed a uniform architecture for the invocation, performance, and closing of exercise sessions. This enables our group or interested external parties to integrate quickly other mathematical systems and develop an appropriate graphical user-interface within the web-browser. We present the details of this generic architecture before presenting the two types of integration that already exist.

The typical flow of messages exchanged during the exercise performance is as follows: In an HTML page of the static book, an exercise is anchored as a URL. It is displayed as an invitation to perform the exercise. A click on its underlying anchor triggers a request to the `ProxyServlet` (a part of the session-manager) which creates the proxy for the given type of exercise. The latter returns the code necessary for the browser to display the user-interface, starts the mathematical system if needed, loads the data of the exercise from the database, and initializes the mathematical system according to this data. The proxy lives till the learner finishes the exercise. At the end, it notifies the user model of the results of the exercise performance. This notification can be used to update the representation of the learner's knowledge and capabilities.

The generic architecture includes proxies and proxy factories. We present shortly the specifications of their classes. These specification allow external developers to create quickly other exercise types. Several such implementations are on their way.

The `ProxyFactory` objects are registered at startup of the system, one instance of such exists per exercise type. They are responsible for:

- return the necessary HTML code to be included into the book for the invitation to perform the exercise
- creating proxies for the type given
- registering and un-registering proxies in the session-manager

One instance of a `Proxy` object exists per exercise performance. It accomplishes the following tasks:

- initializing and connecting to the mathematical system
- provide code to the browser so that it can invoke the user-interface
- possibly transmit messages from the user-interface to the system
- collect results of the exercises and transmit them to the user model

These specifications are general enough to allow the connection of any user-interface. For example, other approaches will certainly also involve running the complete mathematical system on the client, for example interactive systems for elementary geometry.

Moreover, the proxy can be employed for a teacher's connection. For example, a teacher console displaying a clone of the learner's console has been realized for the CAS console applet, see figure 3. This is made possible by the central and accessible position of the proxy.

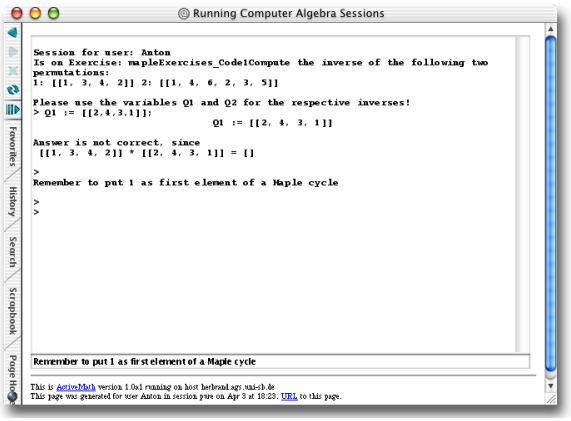

Figure 3: A teacher's *clone console* in ActiveMath observing a learner's activity and sending her suggestions

3.1 Communication Between Components

The XML-RPC protocol has been chosen for the communication between components of the system. XML-RPC requests are structured HTTP POST requests. The content of the request and of the responses are XML encoded messages. . Thus, addresses of servers are simple HTTP URLs which can easily be exchanged as simple strings, for example in applet parameters.

The choice of XML-RPC proved to be particularly appropriate for exercise proxies. Being an extension of HTTP, their access from Java applets is trouble-free. Using these URLs allow the requests to be tunneled through a single HTTP server and enables secure connections and traversing firewalls, two features that are planned for ActiveMath.

XML-RPC is a cross platform standard which is implemented on more than 20 platforms and languages. Its greatest advantage is its simplicity of implementation. For the ActiveMath needs, XML-RPC was implemented for Allegro Common Lisp (for the connection to and from ΩMEGA) and on the Mozart-Oz environment (for MathWeb).

3.2 Connecting to Mathematical Systems

The communication with mathematical systems is typically realized by simple process standard-in and standard-out. Managing the availability and network distributions of such systems is a more delicate task and can be very platform- and installation-specific. For this reason, systems have been conceived to abstract the functionalities of such connections and offer remote access to them through a broker architecture, for instance, MathWeb [6] and, in a more general setting, Corba [13].

The MathWeb broker was originally created as a service-broker to offer computational services to proof-systems. It is implemented in Mozart-Oz [16] which provides complete network transparency. Hence, MathWeb supports the delegation of the request for a non-existing service to some other machines where the service is offered.

Mathematical systems' connection methods are not new and there are several such initiatives, see for example [1] and the references therein. However, one important quality is

rarely found: stateful sessions. Most of the connection methods close the mathematical system and its connection after one computation is finished. This practice is acceptable for the execution of single computations which store the results in another application. It is, however, not suitable for applications that perform several computation steps and for which the intermediate results are required. MathWeb offers stateful session services for the Maple and Gap Computer Algebra Systems as well as for the constraint solver CoSie [12] which are used in ActiveMath.

Thanks to MathWeb, the connection to a mathematical system is made in few lines of code. This is true for most other connection methods as well and ActiveMath will also take advantage of other connection methods. For example the MuPAD Computer Algebra System will be connected through a dedicated java-library for this system.

4 Example Scenarios

We now describe the exercise and exploration mechanisms that have been realized in ActiveMath already. These examples prove that a complete integration into a web-environment can be performed. These examples implement principles of web-integration. They should give a taste of the scenarios that can be deployed for ActiveMath.

Mathematical exercises usually can contain computation and deduction problems. The mathematical systems that support computations are Computer Algebra Systems; deduction problems can be supported by theorem provers and proof planners.

4.1 The Computer Algebra Console

The current user-interface for CAS-exercises is the Computer Algebra console implemented by an applet. The input is in the syntax of the underlying CAS. The philosophy behind this decision is that, in a given course, a learner is interested to learn the syntax of a Computer Algebra System just as she learned operating her desktop calculator a few years ago.

The CAS console applet displays a standard console similar to the Computer Algebra System used in terminal mode. After displaying the necessary instructions to the learner, the console allows input of lines which are executed in the remote CAS. The result of the CAS computation is displayed as well as an evaluation report (see figure 4).

4.1.1 CAS Console Authoring Content

A CAS-exercise is parametrized by a code element stored in the database. This code element consists of three parts in the CAS language: (1) the initialization part which is sent to the CAS on startup and outputs the welcome message to the learner. Typically, it contains load instructions of libraries and definitions of variables. (2) the evaluation part which is executed after each learner input. It checks weather the required definitions have been made by the learner and if so, checks their validity and possibly responds a feedback. Finally, (3) the shutdown part which is executed at the end of the exercise, before the CAS is stopped.

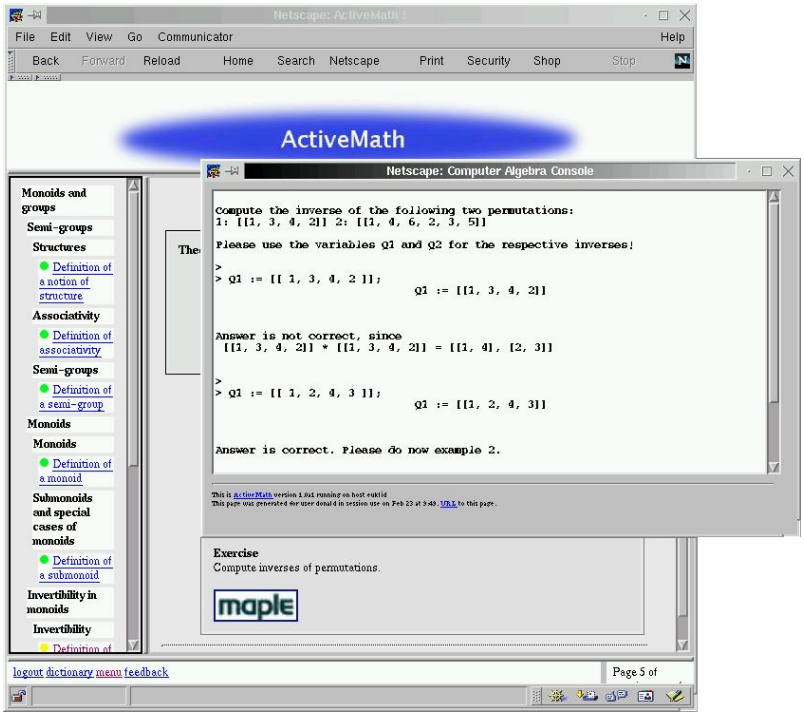

Figure 4: A MAPLE console displaying a simple exercise.

Figure 5 presents an example of a simple code for an exercise in which the learner is requested to compute the inverse of permutations given by cycles. The concrete goal of the exercise is to define, in MAPLE, the variables `Q1` and `Q2` as the inverse of given permutations. The eval-code checks whether these variables are defined and checks if they are the wanted inverses.

```
<code id="mapleExercises_Code1">
<data>
  <startup><command>evalSilent</command><param>
      with(group);
      P1:=[[1,3,4,2]];
      P2:=[[1,4,6,2,3,5]];
    </param></startup>
  <startup><command>eval</command><param>
      printf("Compute the inverse of the following two permutations:");
    </param></startup>
  <startup><command>eval</command><param>
      printf("1:  %a         2:  %a\n",P1,P2);
    </param></startup>
  <startup><command>eval</command><param>
      printf("Please use the variables Q1 and Q2 for the respective inverses!");
    </param></startup>

  <eval><command>eval</command><param>
      if assigned(Q1) and assigned(Q2)
          and evalb(mulperm(P1,Q1)=[])
          and evalb(mulperm(P2,Q2)=[]) then
        1
      elif assigned(Q1) and assigned(Q2) and evalb(Q1=I1) then
        printf("Answer Q1 is correct but not Q2, since\n
            %a * %a = %a", P2, Q2, mulperms(P2,Q2))
      <.....>
      fi;
    </param></eval>
</data>
</code>
```

Figure 5: An example `code` element in the database

4.1.2 Messages Exchanged

The request that creates this CAS console applet is triggered by an anchor on a page of the course that the learner is reading. The request is directed to the `ProxyServlet` which invokes the factory class to create the appropriate proxy. The latter sends an instruction to execute the applet in a new window. Once a CAS-exercise is requested, the `Proxy` requests a MAPLE "service" from the MATHWEB broker and initializes it. In the meantime, the applet is executed on the client-side. This applet is parametrized with the XML-RPC URL that has been given.

Each time the learner sends a command, it is sent to the proxy and executed by the CAS. The response is passed to the applet which can display it. Right after the execution, the proxy also sends MAPLE the evaluation code. When the console window is closed, the proxy can shut down the mathematical system and reports the results to the user model.

4.1.3 Advantages and Drawbacks

The usage of such an applet is quite efficient. A major advantage is that the CAS mathematical capabilities are available to the content writer to have a faithful evaluation of the learner's input. Also, the learner has the complete freedom to use the CAS as an explorative tool as the complete syntax of the CAS is accessible.

The accessibility of the complete syntax for the learner, can, however, raise requirements. For instance, it appears necessary to disable the use a given functionality that makes the exercise too easy to solve. Security issues can also arise as these systems have not been planned for such usages and often offer commands that may be dangerous. To prevent abuses, the learner's input has to be filtered. The console-applet proxy already has built-in such a filtering mechanism. This mechanism will, however, never be complete as the recognition of such commands require a complete understanding of the system's language[2]. To be able to solve this problem entirely, it appears unavoidable to request the support of the CAS, e.g., the MAPLE and MUPAD function redefinition.

4.1.4 Other Usages

The CAS console can also be used in another kind applet which offers the visualization of a phenomenon whose state variables are reflected in CAS variables. We have built an exploratory applet to illustrate the inscribed triangle theorem which you can see in figure 6. For each drags of the vertex A, variables ax, ay, etc, are defined in MAPLE with the coordinates of the points and the angle at A is recomputed. The CAS console applet then allows the learner to invoke computations of her choice using the coordinates of the points that she is viewing. This presents a good illustration of "explorations": such an interactive applet does not require the learner to reach a precise goal (like in an exercise) or to follow a precise track (like in an example).

[2]For example, MAPLE syntax allows the evaluation of an expression created as the concatenation strings.

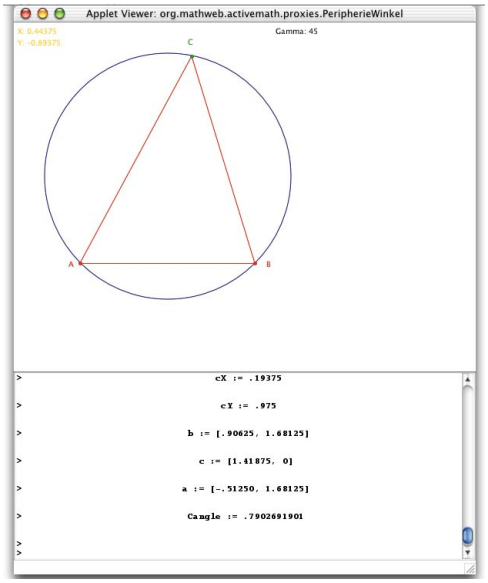

Figure 6: A more graphical interactive usage of the console-applet.

4.2 The ΩMEGA-LΩUI Applet

For deduction problems, that is to say, the construction of proofs, there exist basically two sorts of computer systems: Automated theorem provers, and interactive systems that are used interactively to construct a proof.

The ΩMEGA systems covers both aspects, the automated search for proofs, with automated proof planning [11] and the support of interactive proof plan construction via an agent-based command suggestion mechanism. ΩMEGA's graphical user-interface, LΩUI [15], includes a proof tree presentation, a sequence of formal proof lines as well as a natural language verbalization of the proof.

The current LΩUI is a implemented in Mozart-Oz. It requires installation of the freely available Mozart-Oz-engine. This is not a big problem currently, as the community of users of LΩUI is still very close to automated theorem proving. A re-implementation of LΩUI as an applet, however, is planned.

The learner communicates with LΩUI via a dialog console, where she can select the next subgoal, apply a proof step (methods), instantiate a variable with a term, or undo proof steps. The list of suggested methods is generated by a command suggestion mechanism. This mechanism allows a specification of when a method should appear in the dialog box. This specification may depend on the user model as well as previous (maybe failed) proof attempts. As a result, the system can react to the learner's choice of an inappropriate method.

When the learner is stuck in a proof situation, she can invoke the automatic proof planner that will insert the next steps. Figure 7 shows LΩUI with a graphical representation of the proof tree, the dialog console, and a verbalization of the current proof.

Just as for the Computer Algebra console-applet, a proxy has been implemented for the ΩMEGA-LΩUI applet to perform the exercises: The button to invoke the exercise is displayed

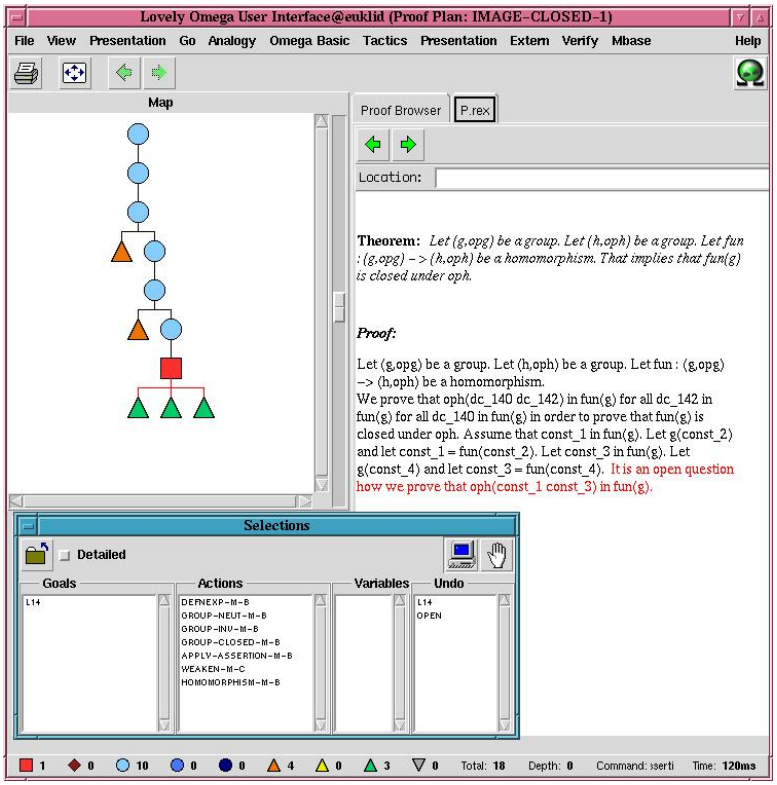

Figure 7: An ΩMEGA-LΩUI session

within the course pages. When invoked, it launches a request to the `ProxyServlet` which delegates the call to the produced `Proxy`. The `Proxy` requests an Oz engine (currently the MATHWEB broker) to prepare an Oz-applet which contains all the parameters of the exercise, including the information to connect to the proxy and to the MATHWEB broker. The request is redirected to this engine and the Oz-applet is sent. MIME-type configurations installed on the browser by the Oz installation launches the Oz virtual machine to open the Oz-applet[3].

The Oz-applet can then start on the client. It requests an ΩMEGA "service" from the MATHWEB broker and tells this service to connect to the proxy. The proxy sends the exercise and the start up-code to ΩMEGA. This content is also encoded in the `OMDoc`s and is extracted from MBASE. Later, when the the learner quits LΩUI, a report is made to the user model and the ΩMEGA server is closed.

5 Enhancing the Integration into ACTIVEMATH

The last section of our article describes future work that is possible thanks to the proxy-based architecture of interactive exercises and thanks to the `OMDoc` encoding of content. The work is pursuing two directions: (1) the integration of the knowledge representation where the flexibility and "polymorphism" of the semantic encoding is used and (2) the enhanced user-adaptivity.

[3]The need for the re-encoding of this applet is that a *document* delivered with MIME-type cannot have parameters.

5.1 Use of Knowledge Representation

As ActiveMath is based on a database of mathematical objects, all of its components can use identifiers referring to these objects. One typical application of this representation is realized in ActiveMath, a dictionary-like to browse the elements. Moreover, an exercise user-interface could *refer* to a definition using its common name and offer the learner the possibility to browse its text.

Other advantages stem from the `OMDoc` encoding of the content. `OMDoc` mathematical expressions are expressed in the `OpenMath` syntax. This syntax is "polymorphic" in the sense that it can be translated to many target languages and, in particular, to the different languages of Computer Algebra Systems. Actually, it has been one of the `OpenMath` major goals to exchange messages between mathematical systems; the conversion between the various syntax is performed by the so-called phrasebooks which are emerging currently.[4]

Remember that the exercise data is loaded by the proxy from the database as part of the initialization process. This initialization is inherently system specific. It needs to contain instructions such as a library to be loaded or internal state modifications such as a function disablement. Mathematical objects, however, e.g., functions, polynomials, groups, or matrices can be encoded in `OpenMath` and converted on the fly. This facility is useful for content authors as it allows them to reuse an `OpenMath` object from the content or to prepare exercises to run with multiple mathematical systems (where only the system-specific parts have to be written for each of them). This `OpenMath` *parametrization* of exercises is currently being implemented.

The same conversion can be used to support the learner performing an exercise. We are planning to implement the drag-and-drop gesture for mathematical symbols from the HTML content pages, in HTML, to the CAS console.

5.2 User-Adaptivity Integration

The initialization process of an exercise can take advantage of the connection to the user model connection. Suppose, an exercise is initialized and this initialization involves the definition of a particular method, operation, or function. The exercise author might decide that the method is only loaded if the learner has sufficient knowledge of it. If the method is loaded, the learner can use it freely. If it is not, however, the only way for the learner to apply the method is to perform the steps of this method manually.

Similarly to the teacher-monitor which can offer external support in the performance of an exercise, a reasonning engine can observe the learner's steps and can perform a more appropriate update of the user model. This usage of the witnessing role of the proxy can be enhanced. For example, such an observing reasoning agent could provide feedback to the learner within the exercise performance.

[4] See, for example the PolyMath Maple phrasebook: `http://pdg.cecm.sfu.ca/openmath/`, the PearlServers implementations for MuPAD and Maple: `http://www.webpearls.com/products/ps.asp`, or the RIACA phrasebooks: `http://crystal.win.tue.nl/projects/index.html`

6 Comparable Solutions

We did not find software that could really be compared to ActiveMath as an integrated platform to learn mathematics. Most of the mathematics published on the Web is encoded as simple LaTeX, DVI, postscript, PDF, or HTML pages (for an overview of these, see for example, the math forum[5], some web-sites seem to use interactive content – mostly dedicated applets – almost none are connected to mathematical systems, very few are adaptive. None seems to encompass a general content encoding that provides re-usability.

A veteran in the field of mathematics on the Web is the Connected Curriculum Project CCP[6] which presents on the Web an introduction to many mathematical subjects together with manipulations on a system and offers the download of a worksheet and detailed instructions for each steps to be performed in the system. CCP is a mature math-publishing project and displays already a lot of content, freely available to interested learners. The comparison to ActiveMath is striking: CCP modules are written in plain HTML. The instructions for the usage of the mathematical system are given step-by-step in a very detailed way, as no communication to it is possible (almost each key is described). Finally no content is dynamically generated and most of the formulas are created as images. As a consequence, the content presented is an immutable collection of books with instructions for the manipulation of a CAS.

Two packages do, however, provide a flexible connectivity to one or more mathematical systems through web-interfaces with simple authoring facilities: WWW Interactive Mathematics Server (WIMS[7]) and Alice In Mathematics (AIM[8]). Both of these tools are freely downloadable and ready to run. Both offer a server solution that connects to a mathematical system. Through an HTML-form-based interface, AIM offers access to automatically corrected exercises or assignments. The system is based on the Maple Computer Algebra System whose language is used to generate exercise parameters on the fly (e.g., bounded random numbers), to evaluate the learner's response, and to compute and record a grade. WIMS has similar interface. This server offers interesting connectivity to many mathematical systems. It also provides a (proprietary) script language that an author can embed into HTML content and is executed on delivery time. This language even contains a syntax for elementary mathematical concepts that it can translate to the systems' syntax.

As opposed to ActiveMath, both, AIM and WIMS, do not provide a more general architecture. For example, none of them supports rendering of mathematical formulas or user-adaptivity (aside of the interactivity of the exercises, of course).

Because of the modularity of the ActiveMath system, developers can integrate mew modules which use the other modules' functionalities rather than having to write these functionalities specialized to the exercise subsystem, as was made in WIMS and AIM. For example multiple-choice-questions are rendered using the same XSL templates as the one used for the rendering of the definitions or exercises.

[5] `http://forum.swarthmore.edu`
[6] `http://www.math.duke.edu/education/ccp/`
[7] `http://wims.unice.fr`
[8] `http://allserv.rug.ac.be/~nvdbergh/aim/docs/`

7 Conclusion

The architecture that we have built provides a simple and open framework for integrating mathematical systems and for for authoring integrated interactive exercises. Statistical software is planned to be connected, as well as the MuPAD Computer Algebra System [17].

The ActiveMath system provides a solution for building re-usable mathematical content to be displayed on the Web with full-fledged interactivity. Far away from anything resembling the management of huge collections of single HTML or other media files, the OMDoc format provides an interesting content encoding that abstracts the semantical meaning of mathematical content and gets rid of anything presentational.

This separation of representation and presentation allows the quality management of what is being displayed in the clients to be performed much more effectively.

This approach allows the tight collaboration between authors, multimedia authors and developers to be loosened and their respective specifications to be facilitated. For example, it leaves the browser dependencies to the XSL writers and applet developers, the connectivity to the server-side developers. and only the content to authors.

References

[1] A. C. Andrew Solomon, Craigh A. Struble and S. A. Linton. The javamath api, an architecture for internet accessible mathematical services. see http://javamath.sourceforge.net, 2001.

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

[3] B. Bloom, editor. *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain*. Longmans, Green, New York, Toronto, 1956.

[4] O. Caprotti and A. M. Cohen. Draft of the open math standard. Open Math Consortium, http://www.openmath.org/, 1998.

[5] B. Char, G. Fee, K. Geddes, G. Gonnet, and M. Monagan. A tutorial introduction to MAPLE. *Journal of Symbolic Computation*, 2(2):179–200, 1986. See also http://www.maplesoft.com/.

[6] A. Franke and M. Kohlhase. System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In H. Ganzinger, editor, *16th International Conference on Automated Deduction*, volume 397, pages 217–221. Springer-Verlag, 1999. See also http://www.mathweb.org/mathweb.

[7] A. Franke and M. Kohlhase. MBase: Representing mathematical knowledge in a relational data base. In F. Pfenning, editor, *Proc. 17th International Conference on Automated Deduction (CADE)*, Lecture Notes on Artificial Intelligence. Springer-Verlag, 2000. See also http://www.mathweb.org/mbase.

[8] M. Kohlhase. OMDoc: Towards an internet standard for the administration, distribution and teaching of mathematical knowledge. In E. R. Lozano, editor, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2000*, LNAI. Springer Verlag, 2001. See also http://www.mathweb.org/omdoc.

[9] P. Libbrecht, E. Melis, and C. Ullrich. Generating personalized documents using a presentation planner. In *ED-MEDIA 2001-World Conference on Educational Multimedia, Hypermedia and Telecommunications*, 2001. accepted.

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

[11] E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 115(1):65–105, November 1999.

[12] E. Melis, J. Zimmer, and T. Müller. Integrating constraint solving into proof planning. In Ringeissen, editor, *Frontiers of Combining Systems, FroCos-2000*, Lecture Notes on Artificial Intelligence. Springer, 2000. `http://www.ags.uni-sb.de/~jzimmer/MelisZimmerMueller.ps.gz`.

[13] Object Management Group. Corba web-site, January 2001. `http://www.omg.org/corba/`.

[14] A. Schoenfeld, editor. *A Source Book for College Mathematics Teaching*. Mathematical Association of America, Washington, DC, 1990.

[15] J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, and V. Sorge. LOUI: A distributed graphical user interface for the interactive proof system OMEGA. In *Proceedings of the International Workshop "User Interfaces for Theorem Provers 1998 (UITP'98)*, Eindhoven, Netherlands, 1998. `http://www.ags.uni-sb.de/~omega/pub/postscript/LOUI98-UITP.ps.gz`.

[16] G. Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer-Verlag, Berlin, 1995. See also `http://www.mozart-oz.org`.

[17] A. Sorgatz and R. Hillebrand. MuPAD - Ein Computeralgebra System I. *Linux Magazin*, (12/95), 1995. See also `http://www.sciface.com`.