Adaptive Access to a Proof Planner*

Erica Melis[1] Andreas Meier[1] Martin Pollet[2]

[1] German Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Germany
melis,ameier@dfki.de
[2] Saarland University, Saarbrücken, Germany
pollet@ags.uni-sb.de

Abstract. Mathematical tools such as computer algebra systems and interactive and automated theorem provers are complex systems and can perform difficult computations. Typically, such tools are used by a (small) group of particularly trained and skilled users to assist in mathematical problem solving. They can also be used as back-engines for interactive exercises in learning environments. This, however, suggests the adaptation of the choice of functionalities of the tool to the learner. This paper addresses the adaptive usage of the proof planner MULTI for the learning environment ACTIVEMATH. The proof planner is a back-engine for interactive proof exercises. We identify different dimensions in which the usage of such a service system can be adapted and investigate the architecture realizing the adaptive access to MULTI.

1 Motivation

So far, the main application of mathematical systems such as computer algebra systems and theorem provers has been for assisting trained and skilled users. This user group determines many design decisions. For instance, interactive theorem proving systems try to support the proof construction by restricting choices to valid proof steps, they suggest applicable lemmas, or they produce a subproof automatically. These functionalities are useful, e.g., for interactively verifying a program and reduce the workload of the proof expert.

Another application of a mathematical system may be as a cognitive tool in a learning environment for which the user group consists of learners rather than proof experts. Empirical evidence indicates that active, exploratory learning helps to construct knowledge and skills in a learner's mind. In particular, empirical studies [13] suggest that student's deficiencies in their mathematical competence with respect to understanding and generating proofs are connected with the shortcoming of student's self-guided explorative learning opportunities

* This publication is partly a result of work in the context of the LeActiveMath and iClass projects, funded under the 6th Framework Programm of the European Community – (Contract IST-2003-507826). The authors are solely responsible for its content. The European Community is not responsible for any use that might be made of information appearing therein.

and the lack of (self-)explanations during problem solving. Such an explorative learning can be supported by tools. For this reason, the web-based adaptive learning environment for mathematics, ActiveMath, integrates problem solving systems as back-engines for interactive exercising.

In an educational context the original features of typical mathematical systems are not sufficient. Rather, an educational context requires additional features for effective learning such as

- adaptivity to the learner,
- feedback on the learners activities,
- possibility to make mistakes.

In order to realize those features and to adapt to the student's context, goals, needs, capabilities, preferences, and previous activities, the setting and the user interface of tool-supported interactive exercising needs to be adaptable to the learner in a pedagogically and cognitively sound way. Thus, for interactive exercising the back-engine has to be extended so that it can process information from ActiveMath's student model and pedagogical knowledge base. Similarly, information about the learner's misconceptions or performance in an exercise should be returned to the student model.

In this paper, we describe a first adaptive integration of the proof planner Multi with the learning environment ActiveMath. In particular, we discuss the different dimensions along which the usage of the proof planner Multi can be adapted. Mostly, we describe personalization dimensions although the same setting can also be used for adaptation of accessibility and customization. We investigate the necessary extensions of Multi and the architecture realizing the adaptive access to Multi.

The paper is structured as follows. We start with preliminaries about ActiveMath and the proof planner Multi, since these might not be known to the gentle reader. Afterwards, we identify adaptivity dimensions of Multi for interactive proof exercises and present the architecture for the adaptive access of Multi. In section 5 we describe the realization of some directions of the adaptivity. Section 6 discusses potential extensions of the current approach and future work. We conclude with a discussion of results and related work.

2 Preliminaries

Empirical results indicate that instruction with proof planning methods, which explicitly encode mathematical steps, can be a learning approach that is superior to the traditional teaching of mathematical proof [7]. This motivates to use proof planning for maths education and the connection of the proof planner Multi with the user-adaptive learning environment ActiveMath.

The following is only a brief introduction to ActiveMath and Multi. For more details on proof planning and Multi the interested reader is referred to [9, 8]. For more details on ActiveMath see [6].

2.1 ActiveMath, its Student Model and Pedagogical Knowledge

ActiveMath is a web-based, user-adaptive learning environment for mathematics. The learner can choose learning goals to achieve a scenario. ActiveMath generates learning material user-adaptively, i.e., dependent on the learner's goals, learning scenarios, preferences, and knowledge. A course generator determines the concepts that the student needs to learn for a goal chosen by the learner and selects the appropriate instructional items (explanations, definitions, examples, exercises, etc). According to pedagogical knowledge, this selection also includes the number, type, and difficulty of exercises and examples, as well as the interactivity setting of exercises (e.g., what is requested from the learner, which functionalities can be used). It assembles the instructional items in a sequence which is suggested by the scenario chosen and the pedagogical strategy following it. The adaptivity is based on a *student model* that includes

- the student's mastery level of concepts and skills
- the history of the learner's actions (e.g., time spent per item)
- her preferences (e.g., preferred language), the chosen learning scenario, and the learning goals as input through a questionnaire.

The student model is updated based on results of the learning activities such as reading and problem solving. That is, the student's exercise performance is evaluated and the evaluation is passed to the student model for updating it.

2.2 Proof Planning

Originally, the goal of the research on proof planning [2] was to prove mathematical theorems automatically. The knowledge-based approach in Multi employs mathematical strategies, methods, and computations for this purpose. It guides the search by heuristics known from mathematical problem solving in specific mathematical areas.

Proof planning starts with a goal that represents the conjecture to be proved and with proof assumptions. It continues by applying a method (such as proof by induction) for which the application conditions are satisfied and this generates new assumptions or reduces a goal to (possibly trivial) subgoals. This process goes on until no goal is left. The resulting sequence of instantiated methods constitutes a solution proof plan.

Generally, proof construction may require to construct a mathematical object, i.e., to instantiate existentially quantified variables by witness terms. In proof planning meta-variables are used as place holders for such objects until enough information is collected to instantiate the meta-variable. A domain-specific constraint solver can help to construct mathematical objects that are elements of a specific domain. During the proof planning process the constraint solver checks the (in)consistency of constraints on meta-variables and collects consistent constraints in a constraint store. Then, it computes instantiations for the meta-variables that satisfy the collected constraints [10].

To structure the repertoire of proof planning methods and make the proof planning process more hierarchical, strategies have been introduced. One of the types of proof planning strategies is specified by a set of methods and search heuristics. Different proof planning strategies can correspond to and implement different proof ideas. In the automatic mode, the proof planner searches for applicable strategies (including object construction and backtracking) or methods in each intermediate state until it reaches a solution proof plan. Mathematics-oriented heuristics guide this search.

For new applications, MULTI has been extended with an interactive mode. In interactive proof planning, the user searches and makes the decisions, which can include the choice of strategies, of methods, and the instantiation of meta-variables.

3 Adaptation to the User

The proof planning service can be requested for proof examples that are (dynamically) demonstrated and for interactive exercises in which MULTI can be used as a back-engine. In these applications, the MULTI service can be adapted according to the learning scenario, the goals, the preferences, and the prerequisite knowledge of the learner. In the following, we shall discuss several adaptation dimensions together with cognitive and other variables of the learner that may affect the adaptation.

3.1 Proof Planning Scenarios

The first dimension of adaptation of the MULTI service is the exercise scenario. There are several types of proof exercises, e.g., those in which the student can interactively apply certain proof planning methods, freely explore, or in which the learner sketches proof steps and MULTI checks the student's steps. Accordingly, we identified several pedagogically motivated proof scenarios. They differ with respect to the overall learning goal and the employed service functionalities. The selection of a scenario is performed by ACTIVEMATH, which requests the MULTI service.

Replay and presentation of a proof plan Completed proof plans (or parts of proof plans) are presented to the learner. The proof plan can be presented at once or stepwise. This scenario targets an understanding of the effects of the application of individual methods, how several method applications combine to a proof plan or provide a basis for self-explaining a proof. The learner's activities in this scenario are mainly restricted to browsing the presentation of the proof plan, of methods and of the constraint store as well as replaying a proof plan step-by-step or hierarchically.

Interactive proof planning The learner constructs a proof plan for a given problem or has to complete a given proof plan with gaps by selecting and applying methods from a pre-defined set of methods or strategies as well as

by instantiating meta-variables. This scenario targets the 'technical' mastery of proof steps and proof heuristics. Moreover, the learner can realize the effects of instantiating a meta-variable and receive support for constructing mathematical objects that instantiate a meta-variable. As we shall see later (see section 5), the learner's main activities are the selection of the next step (and its details) as well as the specification of meta-variables. Other possibilities are browsing the current proof plan and requesting additional information, e.g., from the constraint store.

Island planning The learner constructs a proof sketch for a problem. This scenario targets the understanding of the proof process without details. The student is supposed to find a proof idea and to provide a structure of the proof by specifying important intermediate goals in a proof plan, so-called proof islands. The main user interactions in this scenario are adding proof islands as well as links between the islands, the theorem and the assumptions that describe which proof nodes depend from which other proof nodes. In addition, the current island plan can be browsed and additional information can be requested.

Free exploration The learner obtains full access to the proof planner. She has to state the problem and initiate the proof process. Moreover, she can freely access different kinds of proof manipulations (application of strategies, of methods, of tools, instantiation of meta-variables). This scenario is only sensible for advanced learners. It targets exploration and discovery learning.

Meta-Cognitive Framework Polya suggested a framework for teaching mathematical problem [11]. He formulates a set of heuristics cast in form of brief questions and prompts within a frame of four problem solving stages: (1) Understand the problem (2) Devise a plan (3) Carry out the plan (4) Look back at the solution. Questions and prompts for (2) are, for instance: do you know a related problem? did you use all the data? Following Polya's ideas, each of the above scenarios can be enriched with meta-cognitive scaffolding in form of context sensitive subtitles, questions and prompts. This targets a structured proof process with separated phases.

3.2 Other Adaptation Dimensions

Other important adaptation dimensions control the possible or preferred **problem solving strategies**, the range of method choices, domains from which mathematical objects can be drawn etc. For instance, an exercise about the limit of a sequence can be solved by the application of limit theorems proved before or by using only the definition of the limit. The choice of such a proof idea depends on the learner's capabilities and on the pedagogical strategy.

In the interactive proof planning scenario MULTI provides suggestions of method applications and meta-variable instantiations to the learner (see section 5). In this scenario an adaptation decision is whether all the suggestions have to be correct or not. This is important, since errors can be an important source of learning. As opposed to a learning context **faulty suggestions** are

not appropriate for problem solving assistance. For learning, however, it might not be the best idea to make only correct suggestions because then the student might just click on any suggestion rather than learn anything.

For instance, when the student has to prove the limit of a sequence by applying theorems and she is already familiar with the theorems, then it may be too boring and not interesting to suggest only applicable theorems. Again, the decision on when to make which faulty suggestion depends on the student's situation and on her capabilities and on the pedagogical strategy.

Further dimensions for adaptation are the **user interface appearance** and the **feedback**.

Note that, so far, we focused on the realization of the adaptive application of the interactive proof planning scenario. A description of the adaptivity realized so far is given in section 5. An adaptive graphical user interface and adaptive feedback delivery are not realized so far.

3.3 Some Variables to Adapt to

Adaptation may depend on the learner's expertise. For instance, a student who is a novice in proving will have more difficulties to choose between many methods and therefore a large set of alternative methods to choose from should be avoided because it increases the likelihood of guessing instead of learning and reasoning.

Adaptation may also depend on the learner's activity history. For instance, it seems not to be advisable to make suggestions for a method, in case the student has not been able to apply that method for several times recently.

Many dimensions of adaptation do not only depend on the learner's aptitudes but also on the chosen pedagogical strategy. For instance, the decision when to allow for faulty suggestions will depend not only on the student's situation and on her capabilities but also on the pedagogical strategy.

4 Architecture

For all those adaptations of the proof planning service the actual proof planner has to be extended by a scenario management and by mediators as described below. That is, the service encapsulates mediators which provide a (Web-) communication interface and compute some of the adaptations. The communication relies on XML-RPC protocols[1] and the OMDOC language [3] and is not further considered here.

Figure 1 depicts the architecture of the proof planning service and its communication with the learning environment ACTIVEMATH including its student model and pedagogical knowledge.

The architecture separates different functionalities and different kinds of knowledge. This modularization helps to maintain and update modules and to re-use components. As a side effect, the GUI becomes more independent and

[1] `http://www.xmlrpc.com`

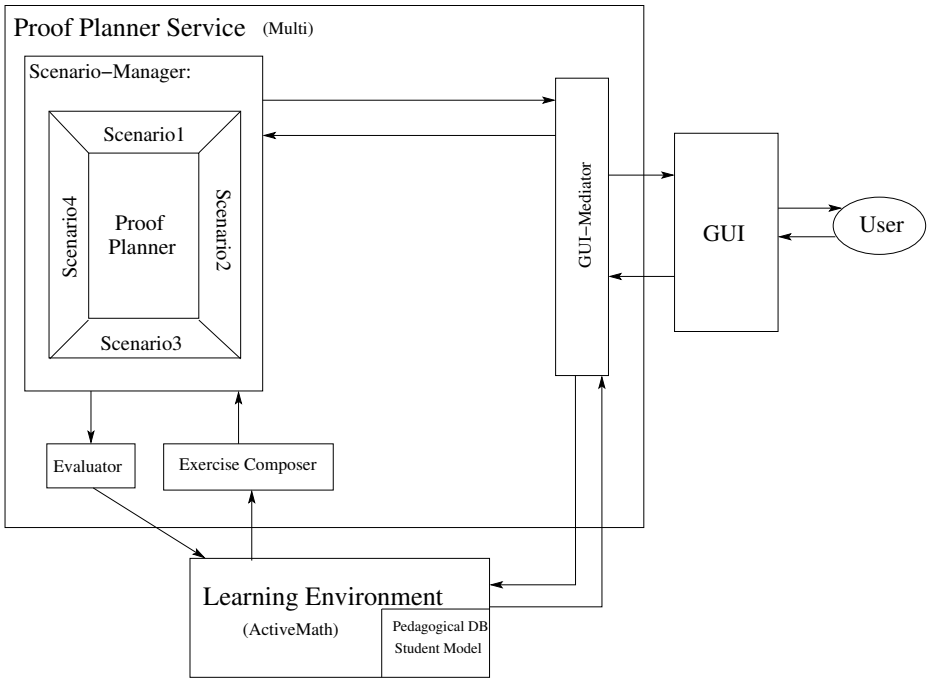

Fig. 1. The architecture of the MULTI service and its communication with ACTIVEMATH and the GUI

adaptable. This part of the architecture (i.e., GUI and GUI-mediator) is, however, not yet implemented.

Scenario-Manager. The Scenario-Manager provides a communication shell around MULTI and realizes the different proof scenarios, which use particular proof storage and proof manipulation functionalities of the proof planner. The scenario for interactive proof planning employs – among others – the following functionalities:

- compute method application and meta-variable instantiation suggestions for the learner,
- check whether method applications and meta-variable instantiations issued by the learner are applicable,
- apply applicable steps,
- check whether proof planner can automatically complete the current proof plan,
- analyze the differences between the current proof plan fragment and a solution proof plan completed by the proof planner.

Relevant for island planning are, in particular, the functionalities:

– apply the island nodes and links specified by the user as abstract steps in the proof plan under construction,
– try to verify islands with automated proof planning or other tool support available in MULTI (e.g., with integrated computer algebra systems or automated theorem provers).

The Exercise Composer and the Evaluator provide interfaces for the communication between ACTIVEMATH and the Scenario-Manager.

Exercise Composer. When ACTIVEMATH requests a proof planning session, then the Exercise Composer computes the input for the parameters of the Scenario-Manager and the scenario. For instance, when the interactive proof planning scenario is requested, then the Exercise Composer determines the range and restriction of suggestions that the scenario component computes and provides to the learner. The Exercise Composer uses student model information for this computation.

Evaluator. At the end of an exercise the Evaluator receives an account on how complete and sound the learner solved the exercise. From this account the Evaluator computes information, which summarizes the learner's actions. The Evaluator passes this information to the persistent student model of ACTIVEMATH.

GUI. The GUI is an independent component. It will be an applet or even a standalone GUI (not integrated into ACTIVEMATH).

GUI-Mediator. The GUI-mediator is needed to adapt the presentation and feedback to the learner's needs and to configure the GUI. The GUI-Mediator realizes the communication between the GUI and the other components of the proof planning service. For each scenario, it interprets the meaning of GUI-objects and their manipulation and passes the interpretation of manipulations to the components of the proof planning service. Vice versa it translates information from the components to the GUI. This is an adaptive translation depending on information from the student model. Its main functionalities are:

– invoke and configure the GUI for a scenario and the individual learner. Adapt the GUI to user characteristics such as preferred language and to the peculiarities of the different scenarios (different scenarios require different user activities and different GUI-objects).
– interpret user interactions with the GUI and pass interpreted information to the Scenario-Manager. For instance, moving a goal-object and connecting it to assumption-objects in the GUI.
– request information/services from ACTIVEMATH, e.g., the explanation of a method or of a definition, as well as from the proof planner, e.g., the current constraint collection, and pass the results to the GUI. This is important, since the learner should be able to actively request different kinds of supporting information.

– interpret proof plan changes, feedback and messages from the proof planner components and pass them to the GUI for display. This includes the change of the proof state as well as proof planning-related messages. For instance, if a method is not applicable because its application conditions are not true, then this feedback can be displayed appropriately because it might help a learner to correct her choice of a method or its details. Since the proof planner generates feedback in a technical format the learner is not supposed to understand, the GUI-Mediator has to interpret and filter this information to provide the learner with useful, comprehensible and possibly adaptive feedback.

5 Adaptive Suggestions of the Glass-Box Service MULTI

The functionalities of black-box services, such as computer algebra systems, are mostly restricted to their original purpose and only few of them can be employed for means of adaptation. For instance, the computer algebra system MAPLE [12] can be called with assumptions for all following computations (e.g., $a > 0$) or certain functions can be excluded from the interaction. The restrictions may not be principled in nature and extensions similar to those described above may accommodate more adaptivity.

Compared with black-box systems the adaptive usage of glass-box systems such as MULTI can be handled more easily. The simple reason is that we have more control over the functions of the glass-box and the extended architecture allows for adaptations. In what follows, we describe some directions of the adaptivity for the interactive proof plan scenario in more detail.

Adapting the Configuration

The list of all parameters of a scenario is called a *configuration*. A configuration for interactive proof planning comprises

– a proof planning strategy,
– a set of suggestion agents,
– the level of automation.

The interactive proof planner comprises an agent-based mechanism for suggesting commands, which specify a method and the arguments necessary for its application to the current proof (see [4] for a detailed description). The set of suggestion agents controls the level of freedom in the interaction with the proof planner. It can be more or less restricted, more or less guided, and it can encode tutorial strategies such as the deliberate introduction of faulty suggestions for learning from failure.

The Exercise Composer computes an initial set of agents from the specifications of methods to be considered for the exercise problem. For instance, one set of agents suggests only applicable methods with all parameters instantiated. Another set of agents suggests only a partially specified method application, which has to be completed by the student.

If the author of an ActiveMath exercise wants to have a particular suggestion, e.g., a suggestion that corresponds to a typical error, then special agents can be added. The Exercise Composer evaluates these additional agents and combines them with the automatically generated ones.

Two other parameters of the interactive proof planning scenario that are determined by the Exercise Composer are 'strategy' and 'level of automation'. Proof planning strategies can implement different ideas for proving. That is, different strategies tackle a proof planning problem in a different way. If the Exercise Composer selects a strategy for the problem at hand, then it computes at least one agent for each method of the strategy. Some of the methods are pretty irrelevant for learning and therefore are applied automatically. The set of all those methods is called the level of automation.

Selection of a Strategy

A (proof) problem may be solvable by different (proof planning) strategies that represent different ideas of how to solve the problem. For instance, the general problem of classifying a residue class structure according to its algebraic properties (associativity, the existence of inverse elements, and isomorphy of residue classes) can be tackled by three different strategies (see [5]): the first strategy tries to solve the problem by applying known theorems, the second strategy reduces a residue class problem to a set of equations, which have to be solved, the third strategy introduces a case split over the (finitely many) elements of the residue class structure.

The Exercise Composer chooses a strategy depending on the concrete problem and on the knowledge of a learner (whether she knows the theorems that are the prerequisites of the first strategy, whether she knows the methods employed by a strategy) and her performance in previous exercises (e.g., when the other strategies have been trained already). Such configuration heuristics can be encoded by pedagogical rules. For instance

```
IF studentKnowledge(prerequisites (firstStrategy)) > medium
AND studentKnowledge(firstStrategy) < medium
THEN present exercise-for(firstStrategy)

IF  studentKnowledge(firstStrategy)  > medium
AND studentKnowledge(secondStrategy) > medium
AND studentKnowledge(thirdStrategy)  < medium
THEN present exercise-for(thirdStrategy)
```

Selection of Agents

If the goal is to most rapidly prove a conjecture and deep learning is unimportant, then the Exercise Composer generates agents for the configuration that check for applicability and provide only fully-specified, applicable suggestions. However, this is not a typical goal in learning to prove. Rather, learning involves to understand why a method is applicable, what a particular method is actually

doing, and for which purpose it is applied. Such competencies may be better trained when input is requested from the student (rather than clicking only) or while making mistakes, discover, and correct them.

One method may correspond to several agents that differ in how specific their suggestions are, i.e., how much input is left to the student. For instance, an agent for a method, which has as arguments a goal and premises, can have suggesting agents for the premises or leave the selection of premises to the learner.

Depending on the student model and the learning goal of an exercise, the Exercise Composer chooses agents for a method that compute more or less complete suggestions. For instance, a novice learner would start with much support and fully-specified suggestions. For a more experienced learner, under-specification can force the learner to specify more input herself in order to discover and overcome misconceptions in the application of a method.

An author-specified agent may request further arguments. For instance, one method for residue class proofs uses a computer algebra system to simplify a modulo-equation. An agent added by the author requests from the student to input the resulting term in advance. During interactive proof planning this input is compared with the actual result of the computer algebra system computation. The idea behind such an agent is to stimulate the anticipatory reasoning of the learner.

Level of Automation

The automation of the application of certain methods avoids bothering the learner with the specification of proof steps, which she already knows. Methods that decompose logical quantifiers and connectives are typical examples for automated methods. Moreover, methods that perform some normalization or re-writing of assumptions and goals can be applied automatically, in case the learner is diagnosed to understand the outcome of these methods.

6 Future Work

6.1 Extensions of the Architecture

The architecture in Fig. 1 (see section 4) enables a "one-shot" adaptive invocation of the MULTI service. That is, when the MULTI service is requested, the architecture enables the selection of a scenario and a configuration depending on the information in ACTIVEMATH. During exercising, the configuration may turn out to be inappropriate for the student. For instance, gaps or faulty suggestions may be to difficult for the student.

In order to allow for adaptation during an interactive exercise we are currently developing and integrating a local student model as well as diagnosis and feedback components into the architecture in Fig. 1.

Local Student Model. The local student model contains information about the user's actions during the proof session and mastery information relevant in that

session. The local student model is maintained during the proof session only. It is initiated by the Exercise Composer with information from ActiveMath's persistent student model. When a proof session terminates, the Evaluator interprets the information in the local student model and passes update information to ActiveMath' persistent student model. The GUI-Mediator creates entries in the local learner history.

Diagnose Component. With the help of Multi the diagnosis component analyzes the interactions of the learner as well as the proof progress during a session. It may use information in the local student model as well as information on the current proof state provided by the Scenario-Manager. The diagnosis component can change the local student model.

Feedback Component. The feedback component uses the diagnosed and collected information in order to compute reactions. This comprises verbal feedback for the user (via the GUI-Mediator), new suggestions, or the modification of the configuration, i.e., the scenario setting, during an exercise.

6.2 Deliver what the Learner Needs

Crucial for the application of Multi for learning is a user-friendly GUI. This adaptive GUI has to hide technical details of the underlying proof engine. A first GUI will be specified based on the outcomes of Human Computer Interaction experiments with different groups of learners. Some features we identified already as crucial are a MathML-like quality of formula rendering, access of subformulas by mouse click, subformula drag&drop, and an input editor.

Beyond the mere presentation in a GUI, for many functionalities in Multi (that can provide important information for an expert) we will have to examine by experiments whether and to which extend they are usable by particular groups of learners. As an example, consider the collection of constraints during the planning process in a constraint store and the application of a constraint solver. The constraint solver provides valuable information for proof planning such as detection of inconsistencies to guide the proof planning and construction of instantiations of meta-variables. For the usage of this information for learning we have to examine questions such as:

- for which learners is the provision of constraints suitable?
- in which form should constraints be shown to particular groups of learners?
- how should/can the consequences of actions to the constraint stores be demonstrated to learners? (e.g., the consequences of an instantiation of a meta-variable by the learner)
- which learners can deal with information such as that their current goal is inconsistent with some collected constraints?
- how should such information be provided to particular groups of learners?
- ...

7 Conclusion and Related Work

We described the architecture for the integration of a cognitive tool in a learning environment. For such a tool to be used for learning, additional components or features may have to be created rather than just a communication wrapper for Web-communication and brokerage.

We have shown how adaptivity can be introduced for a proof planning service in order to adapt to the context and the needs of the learner. The adaptivity is realized by extensions of MULTI as well as by the architecture that uses mediators that evaluate student model and context information to provide adaptive access to MULTI and to a GUI. The architecture emphasizes the separation of different functionalities by different components.

The concrete extensions of MULTI are tool-specific. However, such extensions can be developed for other glass-box systems as well.

For intelligent tutoring further components are necessary. In particular, we are currently designing a local student model and diagnosis and feedback components. Moreover, we started experiments to empirically test which functionalities of proof planning are suited for particular groups of learners and how to provide the functionalities to the learner in a usable way.

Related Work

Aspinall describes in [1] the Proof General Kit, which provides a general architecture for the integration of proof assistants with so-called display engines. The architecture consists of a collection of communicating components centered around the Proof General Mediator, which interfaces all the components and handles the communication between the components. This approach clearly separates the GUI from the proof machinery and enables the integration of different proof assistents into a uniform framework with uniform presentations in different user interfaces. In our architecture, the mediators (the Exercise Composer and the GUI-Mediator) focus on the realization of adaptations.

Acknowledgement We thank numerous members of the ActiveMath-Group for their contributions in discussions.

References

1. D. Aspinall. Proof general kit, white paper. 2002. Available from http://www.proofgeneral.org/kit.
2. A. Bundy. The use of explicit plans to guide inductive proofs. In *Proc. 9th International Conference on Automated Deduction (CADE-9)*, volume 310 of *LNCS*, pages 111–120. Springer-Verlag, 1988.
3. M. Kohlhase. OMDOC: Towards an internet standard for the administration, distribution and teaching of mathematical knowledge. In *Proceedings Artificial Intelligence and Symbolic Computation AISC'2000*, 2000.
4. A. Meier, E. Melis, and M. Pollet. Adaptable mixed-initiative proof planning for educational interaction. *Electronic Notes in Theoretical Computer Science*, 2004. To appear.

5. A. Meier, M. Pollet, and V. Sorge. Comparing approaches to the exploration of the domain of residue classes. *Journal of Symbolic Computation, Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems*, 34(4):287–306, 2002.
6. E. Melis, J. Buedenbender, E. Andres, A. Frischauf, G. Goguadse, P. Libbrecht, M. Pollet, and C. Ullrich. ActiveMath: A generic and adaptive web-based learning environment. *Artificial Intelligence and Education*, 12(4):385–407, 2001.
7. E. Melis, C. Glasmacher, C. Ullrich, and P. Gerjets. Automated proof planning for instructional design. In *Annual Conference of the Cognitive Science Society*, pages 633–638, 2001.
8. E. Melis and A. Meier. Proof planning with multiple strategies. In *First International Conference on Computational Logic*, volume 1861 of *LNAI*, pages 644–659. Springer-Verlag, 2000.
9. E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 115(1):65–105, 1999.
10. E. Melis, J. Zimmer, and T. Müller. Extensions of constraint solving for proof planning. In *European Conference on Artificial Intelligence*, pages 229–233, 2000.
11. G. Polya. *How to Solve it.* Princeton University Press, Princeton, 1945.
12. D. Redfern. *The Maple Handbook: Maple V Release 5.* Springer-Verlag, 1998.
13. K. Reiss, F. Hellmich, and J. Thomas. Individuelle und schulische Bedingungsfaktoren für Argumentationen und Beweise im Mathematikunterricht. In *Bildungsqualität von Schule: Schulische und außerschulische Bedingungen mathematischer, naturwissenschaftlicher und überfachlicher Kompetenzen. Beiheft der Zeitschrift für Pädagogik*, pages 51–64. Beltz, 2002.