Generating Personalized Documents Using a Presentation Planner

Paul Libbrecht, Erica Melis, Carsten Ullrich
DFKI Saarbrücken
D-66123 Saarbrücken
Germany
paul,melis,cullrich@ags.uni-sb.de

The ActiveMath Learning Environment in a Nutshell

ActiveMath (Melis et al., 2001) is a web-based learning environment that dynamically generates interactive mathematical courses adapted to the learners' goals, preferences, capabilities and knowledge. ActiveMath is realized as a client-server web-architecture that can be accessed using standard web-browsers. When learners use the system for the first time, they have to complete a registration form in which they indicate their preferences and abilities and may estimate their mastery levels of the course's domain knowledge. From this information, an inspectable *user model* is generated which is constantly updated when the learner acts in the course. The learner can choose between predefined courses (defined by a teacher) or let the system generate a new course according to the *goal concepts* (mathematical definitions and assertions) and the *scenario* the learner chooses. One possible scenario is *exam preparation* in which only the mathematical definitions of concepts and corresponding exercises are provided. Another scenario is the *guided tour* in which complementary information such as motivating texts, examples and elaborations etc. is selected.

ActiveMath integrates mathematical service systems. Currently, these are Computer Algebra Systems (CAS) and the proof planner of ΩMEGA (Melis & Siekmann, 1999). They can be called to demonstrate an example, to interactively solve an exercise, and to take over certain routine tasks.

The learning material is encoded in the XML-based knowledge representation language `OMDoc` (Kohlhase, 2001). `OMDoc` encodes mathematical objects and items such as definitions, theorems, proof methods, proofs, examples, exercises, and remarks. Each item has its own meta-data that can contain additional information such as dependencies or pedagogical data, e.g., difficulty level or abstractness level of an exercise.

A demo of the ActiveMath system is available at `http://www.mathweb.org/activemath/demo`.

The Presentation Planner

The central component of ActiveMath is the presentation planner. It is responsible for generating a personalized course in a three-stage process:
(1) Retrieval of content. Starting from the goal concepts chosen by the user, all concepts they depend upon and corresponding additional items (e.g., elaborations, examples for a concept) are collected recursively from the knowledge base. This process uses the dependency metadata information contained in the `OMDoc` representation. The result of this retrieval is a collection of all concepts and information that the user needs to learn in order to understand the goal concepts.
(2) Applying pedagogical knowledge. Then, the collection of content items is processed according to the information in the user model and in the pedagogical module. This results in a personalized instructional graph of the learning material. This process is detailed below.
(3) Linearization. Then, the instructional graph is linearized.

The result of the presentation planning is a linearized instructional graph whose nodes are `OMDoc` items. This collection is eventually transformed by XSL-transformations into HTML pages.

Applying Pedagogical Knowledge

The goal of the application of pedagogical knowledge is to select from and transform the collection of items that was gathered in the first stage of presentation planning into learning material. ActiveMath employs pedagogical information represented in pedagogical rules. It evaluates the rules with the expert system shell Jess (Friedman-Hill, 2000). The rules consist of a *condition* and an *action* part. The condition part of a rule specifies the conditions that have to be fulfilled for the rule to be applied, the action part specifies the actions to be taken when the rule is applied.

The presentation planner employs the pedagogical rules to decide: (1) which information should be presented on a page; (2) in which order this information should appear on a single page; (3) how many exercises and examples should be presented and how difficult they should be; (4) whether or not to include exercises and examples that make use of a particular service system. Since the work with service systems requires a certain minimal familiarity with the systems, ActiveMath presents those exercises only if the capability is confirmed. Moreover, pedagogical rules may restrict the available functionalities of a service system. For instance, a student learning about 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.

The application of pedagogical rules works as follows. First, information about the learner retrieved from the user model (e. g. what kind of service system the learner can use) is entered as facts into Jess' knowledge base together with the `OMDoc` items (annotated with the learner's mastery level) collected in the first stage of presentation planning, her goals, and the chosen scenario. Then the rules are evaluated. This results in adding new facts in the knowledge base and eventually generating the sorted lists of items the pages of the course will consist of. In the following, we provide examples of pedagogical rules for two different types of decisions. The rule

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

determines the kind of items and in which order they will appear on the course pages, here for the case that the learner has selected to prepare for an exam (indicated by the fact `(scenario ExamPrep)`). When this rule fires, then the facts `(definition ... exercise)` are asserted, i.e., added to Jess' knowledge base. This implies that these items will appear on a page in the specified order.

In turn, these facts will make other rules fire, e.g., those choosing exercises with an appropriate difficulty level:

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

This rules determines that if exercises should be presented at all (indicated by `(exercises)`) and if there exists a definition d in the knowledge base of Jess, then d's name is bound to the variable `?definition` and the learner's knowledge of d is bound to `?user-knowledge`. Jess allows to specify Boolean functions (indicated by `test`) that are evaluated and whose value determines whether a rule fires or not. The above rule fires when the learner's knowledge is less than 0.3. Then the fact `(choose-exercise-for ?definition (0.3 0.5 0.7))` is inserted into Jess' knowledge base and this triggers the selection of examples for d with difficulty levels 0.3, 0.5, and 0.7 in a row.

References

Friedman-Hill, E. (2000). *Jess, the java expert system shell.* (`http:/herzberg.ca.sandia.gov/jess/`)

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. (forthcoming)

Melis, E., Andrès, E., Franke, A., Frischauf, A., Goguadze, G., Libbrecht, P., Pollet, M., & Ullrich, C. (2001). ActiveMath: A web-based learning environment. *Artificial Intelligence in Education.* (submitted)

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