Course Generation Based on HTN Planning

Carsten Ullrich
German Research Center for Artificial Intelligence, DFKI GmbH
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
cullrich@dfki.de

Abstract

A course generator automatically assembles learning objects retrieved from one or several repositories to a greater unit. Different frameworks for course generation exists, but only a few allow the representation of sophisticated pedagogical knowledge. In this paper, we describe course generation based on pedagogical tasks and methods, formalized in a hierarchical task network (HTN) planner. We describe the moderate constructivist learning strategy we implemented in this framework, problems that arouse from using HTN planning and their solutions. A first evaluation supports the practical value of our approach.

1 Introduction

Course generation automatically assembles learning objects retrieved from one or several repositories to a greater unit, a course. The learning objects are not assembled randomly, but support the user to reach a learning goal.

Such goals require a sophisticated representation and usage of pedagogical knowledge. Although course generation has been a topic of research since long, only a few frameworks exist that offer a representation of pedagogical knowledge powerful enough to encode a variety of generic pedagogical approaches. In this paper, we will describe our approach to course generation, which is based on hierarchical task network (HTN) planning. We will start by reviewing related work (Section 2). Then, Section 3 provides a short introduction to HTN planning. The main part of the paper describes and exemplifies the course generation as developed for LEACTIVEMATH.

2 Related Work

Early approaches on course generation date back to the eighties [Peachy and McCalla, 1986; Murray, 1989]. With the Generic Tutoring Environment (GTE), Van Marcke [van Marcke, 1992; van Marcke, 1998] introduced the separation between instructional tasks (representations of pedagogical activities) and instructional methods (representations of different ways of achieving the activities). Most of this instructional knowledge is independent of the learning domain. Vassileva [Vassileva, 1995] build on this approach and added several layers of rules that handled different aspects of course generation (e.g., selection of the main strategy or of the appropriate media types, etc.).

Subsequent course generation frameworks (e.g., [Specht and Oppermann, 1998; Steinacker *et al.*, 1999]) did not reach such a high level of representing pedagogical knowledge (or at least never provided sufficiently detailed descriptions).

Today's course generation focuses less on pedagogical knowledge, but more on Semantic Web and metadata, e.g., on using ontologies of the subject domain to automatically calculate the best path through the learning material [Karampiperis and Sampson, 2004], or on calculating a learner specific ranking of and trail through learning objects retrieved according to his query [Keenoy *et al.*, 2004]. These approaches use rather simplified pedagogical knowledge, e.g., to select those learning objects with the lowest typical learning time. However, to generate a course which is adapted to the individual learner's goals and needs and which is based on state of the art pedagogical strategies requires more elaborate expertise.

In a former project, we used an expert system (JESS, [Friedman-Hill, 1997]) to represent elaborated course generation knowledge as rules [Ullrich, 2003]. However, this approach had severe scaling problems: in case more than 10 users used the system simultaneously, the latency time increased over an acceptable amount. While technical optimizations might have alleviated this problem, using rules made it difficult to use the hierarchical structure inherent in pedagogical knowledge. Therefore, we decided to investigate a new approach based on HTN planning.

3 Hierarchical Task Network Planning

In HTN planning a planning problem is represented by sets of tasks (*task networks*); *methods* decompose non-primitive tasks into sub-tasks until a level of primitive tasks is reached, which can be solved by *operators* [Erol *et al.*, 1994]. Because HTN incorporates heuristic knowledge in the form of the decomposition methods, it is a very efficient planning technique. It also offers a relatively straight-forward way for representing human expert knowledge. The current version of our course generator uses the HTN planner JShop2 [Ilghami and Nau, 2003], a domain-independent planner that compiles domain descriptions into domain-specific planners.

Figure 1 contains an example of a method (for the time being, we will focus on its HTN related aspects, the underlying pedagogical strategy will be explained in Section 4.1). The method is applicable in case an open task exists that matches with `teachConcept ?c`, with `?c` being a variable (indicated by the prefix `?`). Then, because the method has no preconditions (the empty parentheses in line two), the task will be decomposed in seven subtasks. The subtasks annotated with the prefix `!` are primitive tasks, i.e., tasks that will not be further decomposed.

```
(:method (teachConcept ?c)
         ()
         ((!startSection NewBook ?c)
          (introduce ?c)
          (develop_concept ?c)
          (practice ?c)
          (connect ?c)
          (reflect ?c)
          (!endSection)
         )
```

Figure 1: An HTN method for teaching a concept

3.1 Challenges of HTN Planning

HTN planning faces several difficulties if employed in a Web-based context. However, today most learning environments are Web-based. As an example, the components of the learning environment ACTIVEMATH [Melis *et al.*, 2001], e.g., the learner model, the learning object databases, and learning supporting tools such as a concept map [Melis *et al.*, 2005], are distributed over the Web and available as Web services. Such a setting poses the following challenges:

Distributed Content In a distributed environment, a course generator should be able to integrate content from several databases. This involves the difficulty that traditional AI planning requires evaluating a method's preconditions against the planner's world state. However, this would require mirroring all the content stored in the repositories (or its metadata) in the world state. This is simply infeasible.

Dynamic User Information Similar problems arise with respect to user information. Similar to most systems ACTIVEMATH uses information about the learner stored in a learner model. Part of this information, e.g., the knowledge state, is frequently updated. Thus, the naive solution of adding the learner information in the world state of the planner is unpractical, especially as the learner model can cover a vast range of content and it is unknown exactly which of the information will be relevant.

Learning Services A vast range of services that support the learning process in various ways have been developed, for instance tools that stimulate meta cognition [White and Shimoda, 1999] or perform assessment [Conejo *et al.*, 2004]. A course should integrate these services, not arbitrarily but in a pedagogically sensible way: during the learning process, at specific times the usage of a tool will be more beneficial than at some other time. For instance, reflecting over the learned content may be most effective at the end of a lesson. Additionally, access to these services may vary, depending on the configuration of the learning environment and their general availability. Therefore, the planning needs to take into account dynamic information about these services.

In the following, we will describe how we represent pedagogical knowledge in an HTN planning framework and how we overcome the above limitations.

4 Representing Course Generation Knowledge in an HTN planner

4.1 Pedagogical Background

In LEACTIVEMATH, the University of Augsburg and the DFKI are jointly developing moderate constructivist pedagogical strategies that rely on the "Programme for International Student Assessment" (PISA) framework, [OECD, 1999].

Instead of exclusively assessing curriculum based knowledge, PISA focuses on mathematical literacy and competencies. Mathematical competencies are general mathematical skills such as problem solving, the use of mathematical language and mathematical modeling. In LEACTIVEMATH, authors annotate exercises and examples with the competencies they train (in addition to a subset of standard LOM metadata [IEEE Learning Technology Standards Committee, 2002]). The pedagogical strategies take these competencies into account: first, they ensure that every course contains a diversity of competencies. Secondly, they primarily select exercises of competency levels just outside the learner's current mastery state.

We identified six different pedagogical scenarios, each corresponding to specific needs and high-level learning goals of the user: LearnNew, Rehearse, Overview, trainCompetency, Workbook, and ExamSimulation. For instance, the scenario LearnNew provides the student with an in-depth course that teaches him all necessary material to fully understand the target concepts right from the start. For each scenario, we formalized the pedagogical knowledge required to generate courses supporting the learner to achieve the respective learning goals.

4.2 Course Generation by HTN Planning

Course generation knowledge lends itself to being represented hierarchically. Additionally, as van Marcke [van Marcke, 1992] showed, by representing the pedagogical objectives as tasks and the ways of achieving the objects as methods, sophisticated pedagogical strategies can be encoded. In HTN planning, tasks are a basic principle, too. Therefore, in a first step, we will bring together the notion of pedagogical and HTN task.

A pedagogical task corresponds to the goals a learner wants to achieve. It combines the two dimensions domain (content) and educational goal. Formally, a task t is defined as a tuple $t = (l, c)$, where l is a pedagogical objective and c a unique identifier of a learning object (content goal). While c specifies the concept the course will primarily target, the pedagogical objective determines the kinds of learning objects selected for c.

In JShop, a task is defined as a predicate symbol followed by several arguments. Hence, a pedagogical task can be mapped to a HTN task by mapping the pedagogical objective onto the predicate symbol, and the content goal onto the arguments. Because of this straightforward mapping, in the following we will no longer distinguish between pedagogical and HTN tasks.

A top-level task that serves as a starting point of course generation is `teachConcept id`. The goal of this task is to assemble a structured sequence of learning objects that help the learner to understand the content goal `c`. Using different collections of tasks and methods (i.e., tutorial strategies), this task can be planed differently. Hence, the task-based approach can serve to represent a variety of pedagogical strategies.

Figure 1 illustrates a method developed in LEACTIVE-MATH based on pedagogical principles. It describes how the task of teaching a concept is decomposed in the scenario LearnNew. This decomposition follows a course of action in a classroom which implements constructivist elements and distinguishes between different stages that occur while learning a new concept. Similar solutions were developed for other high-level learning goals.

`!startSection` and `!endSection` are primitive tasks, which serve to provide additional structure. They open and close sections within a course. The introduction (`introduce ?c`) makes the aims of the learning process explicit. It provides an overview to the learner and makes the learning process more transparent. The stage of developing a concept serves to provide the essential information about the concept to the learner, and will be explained in more detail in the next paragraph. The fourth subtask of teaching a concept trains the application of the concept. The final stages, connect and reflect, serve metacognitive purposes. They support the learner to put the concept in its context with respect to the other learning material, and help him to get an overview of his knowledge state.

As a detailed example, consider the two methods for developing a concept shown in Figure 2. The upper method uses information about the user that is retrieved from a learner model and is applicable if the learner exhibits a high math anxiety. In that case, first the concept is inserted in the course via the primitive task `!insertElement ?c`. Then, the insertion of an explanation and an example illustrating the application of the mathematical concept aim at alleviating the anxiety. The bottom method is applied in case the first method does not match, and just inserts the concept itself.

```
(:method (develop_concept ?c)
         ((learnerProperty anxiety ?an)
          (>= ?an 3))
         ((!insertElement ?c)
          (explain ?c)
          (examplify ?c)))

(:method (develop_concept ?c)
         ()
         ((!insertElement ?c)))
```

Figure 2: Two methods for developing a concept

4.3 Critical and Optional Tasks

Course generation has to distinguish between critical and optional tasks. Critical tasks represent necessary elements of the pedagogical strategy that have to be fulfilled for the course generation not to fail. For instance, in a problem-based approach it is mandatory to present a learning object is of the type `realWorldProblem`, hence the corresponding task `insertProblem!` is marked as critical (indicated by the suffix `!`). In contrast, optional tasks should be achieved if possible, but failing to do so will still result in a course. As an example, the subtask `explain` of the upper method in Figure 2 is optional, e.g., if there are no learning object that can serve to explain concept `c` the concept will still be considered as developed.

4.4 Ideal and Fallback Methods

In practice, it is impossible to encode course generation knowledge that covers every potential combination of information about the learner. Oversimplified, if a learner model represents n different kinds of information with m possible values each, then the number of possible combinations is n^m. Writing rules or methods for each case would keep a lot of teachers busy for a long time.

```
(:method (selectAppropriateExercise ?c)
  ;; ideal method: if motivation is
  ;; high, then select exercise with
  ;; higher competency level.
  (;; preconditions:
   (learnerProperty field ?field)
   (learnerProperty educationalLevel ?el)
   (learnerProperty motivation ?m)
   (>= ?m 3)
   (learnerProperty competencyLevel ?c ?cl)
   (try-all-assign ?exercise
       (call GetElements
        ((class exercise)
         (relation for ?c)
         (property learningcontext ?el)
        (property competencylevel (+ 1 ?cl))
         (property field ?field)))))
  (;; sub-task:
   (insertElement ?exercise)))

(:method (selectAppropriateExercise ?c)
  ;; alternative method:
  ;; select exercise from any field
  ((learnerProperty allowedEducLevel ?el)
   (learnerProperty competencyLevel ?c ?cl)
   (try-all-assign ?exercise
       (call GetElements
        ((class exercise)
         (relation for ?c)
        (property learningcontext ?el)
        (property competencylevel ?cl)))))
  ((insertElement ?exercise)))
```

Figure 3: Selecting an exercise

Therefore, in LEACTIVEMATH, methods encode best possible and alternative ways of selecting learning objects. An *ideal* method typically takes a larger number of user properties into account and poses a larger number of constraints on the learning object. As an example, consider the methods in Figure 3. They encode the pedagogical knowledge about exercise selection, and are called as a sub-task of `practice`. The upper method checks whether the user is highly motivated. Then, it searches for an exercise that would be slightly too difficult under normal circumstances (competency level plus 1). Additionally, the learning context of exercise should correspond to the educational level of the learner, and, similar, its field should correspond to the field of the learner.

In case no ideal method is applicable, *fallback* methods come into play. They encode the least constraining conditions on the learning objects possible and serve to insert elements in case no ideal method can be fulfilled. The bottom method of Figure 3 matches if any exercise for concept $?c$ exists that is of the correct competency level and has a learning context equal or lower as the educational level of the learner.

4.5 The Output of Course Generation

The result of the planning is a sequence of learning objects called *course structure*. Similar to the `organization` element of an IMS Content Package [IMS Global Learning Consortium, 2003], it consists of nested sections with the leaves being pointers to learning objects.

Because tasks represent a vast range of pedagogical goals, the size of the generated courses ranges from a single element to a complete curriculum. For instance, while the task `teachConcept` results in sequences of several learning objects, other methods may select only a single element. Frequently occurring examples are the methods for exercise selection shown in Figure 3 and for example selection, shown in Figure 4.

The example selection is additionally interesting because of its differences to exercise selection. Examples serve to increase motivation by presenting the application of a concept. Therefore, the primary requirement is that the exemplary application is taken from the field of interest of the learner, even in case its assigned competence level is higher than the current competence level of the learner.

```
(:method (selectAppropriateExample ?c)
  ;; ideal method: if motivation is
  ;; low, select example with
  ;; matching field.
  ( (learnerProperty field ?field)
   (learnerProperty educationalLevel ?el)
   (learnerProperty motivation ?m)
   (call < ?m 2)
   (learnerProperty competencyLevel ?c ?cl)
   (equivalent ?cl ?ex_cl)
   (try-all-assign ?example
       (call GetElements
        ((class example)
         (relation for ?c)
         (property learningcontext ?el)
         (property competencylevel ?ex_cl)
        (property field ?field)
         ))))
  ((insertElement ?example)))

(:method (selectAppropriateExample ?c)
  ;; fallback method: select example
  ;; from any competency level.
  ( (learnerProperty allowedEducLevel ?el)
   (learnerProperty field ?field)
   (try-all-assign ?example
       (call GetElements
        ((class example)
         (relation for ?c)
         (property learningcontext ?el)
         (property field ?field) ))))
  ((insertElement ?example)))
```

Figure 4: Selecting an example

4.6 Solutions to the Challenges of HTN Planning

In Section 3.1 we described the problem that traditional AI planning requires evaluating a method's preconditions against the planner's world state, which is practically impossible in case the learning objects are stored in one or several databases. Therefore, we extended the HTN planner so that queries about learning objects in a method's preconditions result in a call to a mediator.

Mediators are well known services used in information integration (for a recent overview on this topic, see [Doan *et al.*, 2004]). They act as links between the knowledge processing and knowledge storing components, and offer a uniform query interface to a multitude of autonomous data sources. The LEACTIVEMATH mediator passes incoming queries to the available databases and combines the results.

JShop2's `call` command allows integrating external function calls during planning. The function call `GetElements conditions` (e.g., in Figure 4) queries the mediator to search the repositories for elements that fulfill the given conditions. `try-all-assign ?p ?identifiers` generates all bindings of `?p` to the elements of the returned identifier list. Thereby subsequently all possible values can be tried when backtracking.

Similarly, preconditions involving information about the learner result in queries to the learner model. These queries are encapsulated in a planning axiom as shown in Figure 5. Axioms are used to infer preconditions not directly asserted in the world state. In the figure, the result of the query is bound to the variable `value` using the additional axiom `same`. The axiom `same` returns `true` in case both provided parameters are instantiated and equal, and false otherwise. If one parameter is uninstantiated, the axiom assigns the value of the instantiated parameter to the uninstantiated parameter.

```
(:- (learnerProperty ?property ?value)
    (same ?value (call queryLM ?property)))
```

Figure 5: Querying the learner model

The integration of learning services happens by *service-adding* methods. These methods encode the pedagogical knowledge at what time in the course the learner should use a tool and insert calls to the tool at the appropriate place in the course structure. Later, when the learner navigates through the course, an invitation to use the service is presented to her. Checking for the availability of a service is done by introducing an external function, too. Thereby a method's preconditions can test for the availability of a service. In case the service is available, its call is inserted in the course. Otherwise, an alternative method is applied. This way, the pedagogical knowledge remains reusable, regardless of the actual configuration of the learning environment.

```
(:method (assess)
  ;; if Siette is available,
  ;; then use it for assessment
  ((call ServiceAvailable Siette))
  ((!insertServiceCall Siette)))

(:method (assess)
  ;; fallback: insert some exercises
  ()
  ((insertText assessment)
   (insertExercises 5)))
```

Figure 6: Integration of an assessment service

Figure 6 illustrates the integration of the assessment tool Siette [Conejo *et al.*, 2004]. The upper, ideal method checks whether Siette is available and if so inserts a call to the service in the course structure. In the fallback case, the assessment will be simulated manually. A short text is added in the course that describes that the following exercises will assess his knowledge, and five exercises are inserted.

5 First Results and Conclusions

The results reported in this section focus on technical aspects. Evaluations regarding learning gains are underway and will be reported at a later time.

We performed a first comparison between the old, expert system based (EXSCG) and the new course generator (HTNCG). The tests and all necessary components (ACTIVEMATH server and a database) ran on a 2.8GH Intel Pentium 4 CPU with 2GB RAM. Both course generators repeatedly generated an average length course about the mathematical concept *derivation*.

EXSCG showed a slightly slower performance for single user course generation (average time of 23 and 20 seconds). However, the EXSCG course generation did not terminate in case of the simultaneous generation of more than 20 users. HTNCG slowed down, but still terminated (average of 2 minutes).

We suspect the differences to be caused by the on-demand retrieval of the metadata. In EXSCG, the metadata of the complete content was loaded in the fact-based of the expert system and then the rules were evaluated. In HTNCG, only content relevant for the current plan was retrieved if necessary, i.e., when queried.

It is interesting that the new approach is faster even though an additional indirect step via the mediator was inserted. In the future, we plan to extend the mediator with caching possibilities. First of all, it can optimize queries and secondly avoid queries over the Web.

An additional evaluation compared an XML-RPC based connection and a direct connection between the mediator and one of our database (based on Lucene [Foundation, 2003]). Because of the overhead of processing XML-RPC, we expected the direct connection to outperform XML-RPC. Surprisingly, the direct connection proved not to be much faster. A possible cause may be the limited number of queries (about 270). We plan to further investigate on this topic.

However, a significant speed-up was achieved by using a cached version of Lucene database. There, in case the very same query is posed more than once, the result elements were not retrieved from index but directly from memory. Course generation for a single user decreased to the average time of three seconds, which is about factor 7 faster than compared to the non-cached version. The speedup holds in case of 30 user planning, which took about 20 seconds. This speedup is even more surprising in regard of the fact that due the early stage of implementation the cache was local to a single user, which means instead of one "global" cache, the system used up to 30 "local" caches. We expect the speedup to be caused mainly be the large amount of similar queries, which occur even in a single course generation. For instance, if several examples for the same concept are inserted in a course, a large number of the same queries are processed.

In this paper, we described how HTN planning can be used for representing and executing course generation knowledge that is based on tasks and methods. Besides the described advantages, we plan to use tasks to describe the service offered by a course generator Web service. Additionally, they may be used for communication between user and system, for instance by providing the user the possibility of executing tasks on demand with respect to a given course and thereby intelligently extending the course. Investigating these possibilities will be our next work.

Acknowledgments

The work described in this publication is implemented in the learning environment ACTIVEMATH [1]. ACTIVEMATH serves as the technical basis used in the LEACTIVEMATH project [2], where a consortium of eight European partners is developing an innovative third generation eLearning system for high school, college, and university level classrooms. The system adapts to the learner and learning context and comprises personalization, tutorial dialogues, open student modeling and interactivity that is tool-supported for active and exploratory learning.

The LeActiveMath project is funded under the 6th Framework Programm of the European Community - (Contract Nr IST-2003-507826). The author is solely responsible for its content, it does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of data appearing therein.

References

[Conejo *et al.*, 2004] R. Conejo, E. Guzman, E. Millan, M. Trella, J. L. Perez de-la Cruz, and A. Rios. Siette: A web-based tool for adaptive teaching. *International Journal of Artificial Intelligence in Education (IJAIED 2004)*, 14:29–61, 2004.

[Doan *et al.*, 2004] AnHai Doan, Natalya F. Noy, and Alon Y. Halevy. Introduction to the special issue on semantic integration. *SIGMOD Rec.*, 33(4):11–13, 2004.

[Erol *et al.*, 1994] K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1123–1128, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[Foundation, 2003] The Apache Software Foundation. Jakarta lucene, July 2003.

[Friedman-Hill, 1997] E. Friedman-Hill. Jess, the java expert system shell. Technical Report SAND98-8206, Sandia National Laboratories, 1997.

[IEEE Learning Technology Standards Committee, 2002] IEEE Learning Technology Standards Committee. 1484.12.1-2002 IEEE standard for Learning Object Metadata, 2002.

[Ilghami and Nau, 2003] Okhtay Ilghami and Dana S. Nau. A general approach to synthesize problem-specific planners. Technical Report CS-TR-4597, Department of Computer Science, University of Maryland, October 2003.

[IMS Global Learning Consortium, 2003] IMS Global Learning Consortium. IMS content packaging information model, June 2003.

[Karampiperis and Sampson, 2004] P. Karampiperis and D. Sampson. Adaptive instructional planning using ontologies. In *Proc. of the 4th IEEE International Conference on Advanced Learning Technologies, ICALT 2004*, pages 126–130, 2004.

[Keenoy *et al.*, 2004] K. Keenoy, A. Poulovassilis, V. Christophides, P. Rigaux, G. Papamarkos, A. Magkanaraki, M. Stratakis, N. Spyratos, and P. Wood. Personalisation services for self e-learning networks. In

[1]http://www.activemath.org
[2]http://www.leactivemath.org

N. Koch, P. Fraternali, and M. Wirsing, editors, *Proc. of 4th International Conference on Web Engineering*, volume 3140 of *Lecture Notes in Computer Science*, pages 215–219. Springer, 2004.

[Melis *et al.*, 2001] E. Melis, E. Andrès, J. Büdenbender, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. ACTIVEMATH: A generic and adaptive web-based learning environment. *International Journal of Artificial Intelligence in Education*, 12(4):385–407, 2001.

[Melis *et al.*, 2005] E. Melis, P. Kärger, and M. Homik. Interactive concept mapping in ACTIVEMATH. In *Delfi 2005*, 2005.

[Murray, 1989] W. R. Murray. Control for intelligent tutoring systems: A blackboard-based dynamic instructional planner. In D. Bierman, J. Breuker, and J. Sandberg, editors, *Proc. 4th International Conference of AI and Education*, pages 150–168, Amsterdam, Springfield VA, Tokyo, 1989. IOS.

[OECD, 1999] OECD, editor. *Organisation for Economic Co-operation and Development: Measuring Student Knowledge and Skills – A New Framework for Assessment*. 1999.

[Peachy and McCalla, 1986] D. R. Peachy and G. I. McCalla. Using planning techniques in intelligent tutoring systems. *International Journal of Man-Machine Studies*, 24:77–98, 1986.

[Specht and Oppermann, 1998] M. Specht and R. Oppermann. ACE - adaptive courseware environment. *The New Review of Hypermedia and Multimedia*, 4:141–162, 1998.

[Steinacker *et al.*, 1999] A. Steinacker, C. Seeberg, K. Reichenberger, S. Fischer, and R. Steinmetz. Dynamically generated tables of contents as guided tours in adaptive hypermedia systems. In *Proceedings of the EdMedia & EdTelecom*, pages 167–175, June 1999.

[Ullrich, 2003] C. Ullrich. Pedagogical rules in ACTIVEMATH and their pedagogical foundations. Seki Report SR-03-03, Universität des Saarlandes, FB Informatik, 2003.

[van Marcke, 1992] K. van Marcke. Instructional expertise. In C. Frasson, G. Gauthier, and G.I. McCalla, editors, *Proceedings of the Second International Conference on Intelligent Tutoring Systems*, number 608 in Lecture Notes in Computer Science, pages 234–243. Springer, 1992.

[van Marcke, 1998] K. van Marcke. GTE: An epistemological approach to instructional modelling. *Instructional Science*, 26:147–191, 1998.

[Vassileva, 1995] J. Vassileva. Dynamic courseware generation: at the cross point of CAL, ITS and authoring. In *Proceedings of ICCE'95, Singapore*, pages 290–297, December 1995.

[White and Shimoda, 1999] B.Y. White and T.A. Shimoda. Enabling students to construct theories of collaborative inquiry and reflective learning: Computer support for metacognitive development. *International Journal of Artificial Intelligence in Education*, 10:151–182, 1999.