Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Pedagogically founded courseware generation based on HTN-planning

Carsten Ullrich [a,*], Erica Melis [b]

[a] Dept. of Computer Science and Engineering, Shanghai Jiao Tong University, 1954 Hua Shan Road, 200030 Shanghai, China
[b] DFKI GmbH, Stuhlsatzenhausweg 3, 66123 Saarbruecken, Germany

ARTICLE INFO

Keywords:
Technology-supported learning
Courseware generation
Adaptivity
Pedagogical knowledge

ABSTRACT

Course generation enables the automatic assembly of course (ware) adapted to the learner's competencies and learning goals. It offers a middle-way between traditional pre-authored "one-size fits all" courseware and manual resource look-up. Such an assembly requires pedagogical knowledge, represented, for instance, as expert system rules or planning operators. In this article, we describe the course generation framework Paigos that enables the formalization and application of complex and realistic pedagogical knowledge. Compared to previous course generation approaches, it generates structured courses that are adapted to a variety of learning goals and to the learners' competencies. We describe basic operators and methods for course generation, which are used to formalize seven different types of courses. Furthermore, we present the results of technical evaluations that investigated the performance of the proposed framework. Among others, the results show that Paigos takes less than a second to generate complex courses with an average learning time of about twelve hours. The results also show that large amounts of educational resources are required for an adaptive pedagogically founded course generation.

1. Introduction

Course generation uses information about learning objects, the learner and his learning goals to automatically generate an adapted sequence of resources that supports the learner in achieving his goals. Course generation offers a middle-way between pre-authored "one-size fits all" courseware and individual look-up of learning objects: on the one hand, pre-authored courses will never do justice to every individual learner since it is practically impossible to cater for individual learning goals and characteristics by providing manually authored courses. On the other hand, searching for resources on your own requires a very competent and self-organized learner who is able to assess and to structure the retrieved content. In particular, low-achieving students rarely possess these skills – actually, several empirical studies show that these learners benefit from content organized form them according to pedagogical principles (OECD, 2003; Prenzel, Carstensen, & Ramm, 2004).

However, existing course generators cannot handle *complex* learning goals nor do they generated structured courses. In most previous course generators the learning goal consists of the target concepts the learner wants to learn about, for instance in mathematics "the definition of the derivative function". Then the course generator retrieves all necessary prerequisite resources. But during learning, a user will have different objectives and requires different resources. For instance, when the content is unknown to him, he requires detailed, comprehensive information. Later, he might want to rehearse the content, which requires different content.

In this article, we describe Paigos, a course generator used in the Web-based learning environment ActiveMath (Melis et al., 2006). Paigos advances the state of the art of course generation by using novel techniques from the (Semantic) Web, Artificial Intelligence and technology-enhanced learning.

The article is structured as follows. In Section 2, we define the basic terminology relevant for this article, followed by a brief description of *Hierarchical Task Network Planning* (HTN-planning), which is used for formalizing the pedagogical knowledge. The subsequent sections describe the main contributions of this work. Section 3 introduces the course generation planning domain and contains detailed descriptions of axioms, operators and methods that together realize basic building blocks for course generation. Then, in Section 4 we show how complex course generation scenarios can be assembled by using this basic knowledge. Each of these scenarios caters for a specific high-level learning goal, e.g., discovering new content or rehearsing previously learned material. Section 5 discusses technical evaluations of Paigos. The figures prove that course generation based on non-trivial pedagogical knowledge results in huge amounts of queries to learning object repositories (e.g., a course that consists of about 360 educational resources requires about 100000 queries). The results also show that techniques we introduced with Paigos (such as dynamic subtask expansion) reduce this amount of queries significantly, resulting in planning times of less than a second.

* Corresponding author.
E-mail addresses: ullrich_c@sjtu.edu.cn (C. Ullrich), erica.melis@dfki.de (E. Melis).

2. Preliminaries

2.1. Terminology

Following Wiley (2000), we define learning objects as "any digital resource that can be reused to support learning". *Automatic* (re-) use of learning objects requires a more narrow definition that also addresses how to locate a learning object and granularity. Thus in the scope of this article, instead of learning object, we will use the term *educational resource*, with the following characteristics: an educational resource is an *atomic*, *self-contained* learning object that is uniquely *identifiable* and *addressable* by an URI, i.e., an Uniform Resource Identifier.

To put it differently, educational resources are entities that can be presented in a Web-based learning environment (and thus must be identifiable and addressable). In the following, the term "resource" is often used as a short form of "resource identifier".

Learning consist of more than reading and experiencing educational resources. Usage of tools such as concept mapping tools, graph plotters, etc, is an essential part of the learning process. Ideally, a course generator includes opportunities to use such tools at appropriate moments. In the following, a *learning-support tool* is any application that supports the learner during his learning process in a targeted way and can be integrated into the learning process automatically (a related concept is "cognitive tool", (Mayes, 1992)).

2.2. Course generation

Brusilovsky and Vassileva (2003) distinguish between *course (ware) generation* and *course(ware) sequencing*. Course generation generates a structured sequence of learning objects that is adapted to the learner. The course is generated before it is presented to the learner. This pre-usage generation has the advantage that the course can be shown to the learner, so that its structure is conveyed.

Course sequencing dynamically selects the most appropriate resource at any moment. Thus, the course is not generated beforehand but step-by-step. The benefit of this approach is that it can react to the current context and thereby circumvent problems that arise in course generation if assumptions about the learner change. However, this local approach makes it hard to convey information about the structure of a course and the sequence from start to end can not be presented to the learner. In this article, we focus on course generation with a extension for local sequencing.

Generally speaking, course generators consist of a *domain model* that represents the content to be taught (typically modeled by a directed graph). Metadata provide additional information about the resources (e.g., difficulty level, typical target audience). Ideally, every resource has a type that specifies its instructional role, e.g., whether it is a definition, example, exercise, etc. (Ullrich, 2005). Based on observations of the user's interactions, the *learner model* stores, infers and updates information about an individual user (Brusilovsky & Millán, 2007). The *teaching model* contains the pedagogical knowledge how to assemble a course by using the information provided by the domain and learner model.

2.3. Hierarchical task network planning

HTN-planning is a practically oriented type of AI planning with applications in various domains such as evacuation planning, software systems integration or project planning (Nau et al., 2005). Briefly speaking, in HTN-planning, the goal of the planner is to achieve a list of tasks, where each task is a symbolic representation of an activity to be performed. The planner formulates a plan by decomposing these tasks into smaller and smaller subtasks until *primitive* tasks are reached that can be carried out directly. Sacerdoti (1975) and Tate (1977) developed the basic idea in the mid-1970s.

The course generator described in this article uses the HTN-planner JSHOP2 (Ilghami & Nau, 2003). JSHOP2 is a member of the SHOP2 planner family (Nau, Muñoz-Avila, Cao, Lotem, & Mitchell, 2001). Unlike most other HTN planners, SHOP2 planners decompose tasks into subtasks in the order in which the tasks will be achieved in the resulting plan. This search-control strategy is called *ordered task decomposition*. As a result of this strategy, the current state is known in each step of the planning process (Nau et al., 2005). This allows incorporating sophisticated reasoning capabilities into the planning algorithm, such as calling external functions, which can perform complex calculations or access external information sources.

The JSHOP2 formalism uses different kinds of symbols: variable symbols, constant symbols, predicate symbols, function symbols, compound task symbols, primitive task symbols, and name symbols. All these sets are mutually distinct. To distinguish among these symbols, we will use the following conventions:

- variable symbols begin with a question mark (such as *?x*);
- primitive task symbols begin with an exclamation point (such as *!unstack*);
- constant symbols, predicate symbols, function symbols, and compound task symbols begin with a letter;
- square brackets indicate optional parameters or keywords.

A term is any one of the following: a variable symbol; a constant symbol; a name symbol; a number; a *list term*; a *call term*. A list term is a term of the form $(t_1 t_2 \cdots t_n)$ where each t_i is a term.

A call term is an expression of the form $(call\, f\, t_1 t_2 \ldots t_n)$ where f is a function symbol and each t_i is a term. A call term tells JSHOP2 that f is an attached procedure, ie, that whenever JSHOP2 needs to evaluate a structure where a call term appears, JSHOP2 should replace the call term with the result of applying the external function f on the arguments $t_1, t_2, \ldots,$ and t_n. In JSHOP2, any Java function can be attached as a procedure, as long as it returns a term as a result.

A *logical atom* has the form $(p\ t_1 t_2 \cdots t_n)$ where p is a predicate symbol and each t_i is a term. For instance, the fact that a resource r was inserted in course is represented by (*inserted r*). Logical atoms can be combined to logical expressions using the standard logical operators: *conjunction*, *disjunction*, *negation* and *implication*.

A *task atom* is an expression of the form $(s\ t_1 t_2 \cdots t_n)$ where s is a task symbol and the arguments $t_1 t_2 \cdots t_n$ are terms. The task atom is *primitive* if s is a primitive task symbol, and it is *compound* if s is a compound task symbol. For instance, the primitive task (*!insert r*) represents the goal that a resource r should be inserted into a course. The compound task (*rehearse r*) represents that a course should be generated that supports the learner is rehearsing r.

An HTN-planning problem consists of an *initial state* (represented as a set of *logical atoms* that are assumed to be true at the time when the plan executor will begin executing the plan), the *initial task network* (a set of tasks to be performed), and a *domain description* that contains

Planning operators describe various kinds of actions that the plan executor can perform directly. These operators have preconditions, add and delete lists. An instantiated operator carries out the primitive task associated with it and changes the world state upon its execution according to its add and delete lists. An operator has the form (*:operator h P D A*) where h is the *head*; P is the *precondition*; D is the *delete list*; A is the *add list*.

Methods describe various ways of decomposing non-primitive tasks into subtasks. These are the "standard operating procedures" that one would normally use to perform tasks in the domain. Each method may have a set of constraints that must be satisfied in order for the method to be applicable. A method is a list of the form (*:method h* L_1T_1 $L_2T_2 \cdots L_nT_n$) where *h* is called the method's *head*; each L_i is called a *precondition* for the method; each T_i is called a *tail* or the *subtasks* of the method. In the following, we will often refer to a method by its head only.

Axioms are Horn-clause like statements for inferring conditions that are not mentioned explicitly in world states, but can be inferred from the world state.. An axiom is a expression of the form (*:−a* $L_1L_2 \ldots L_n$), where the axiom's *head* is the logical atom *a*, and its *tail* is the list $L_1L_2 \ldots L_n$, and each L_i is a logical precondition. The intended meaning of an axiom is that *a* is true if L_1 is true, or if L_1 is false, but L_2 is true, ..., or if all of L_1,L_2, ... ,L_{n-1} are false but L_n is true.

Planning is done by applying methods to non-primitive tasks to decompose them into subtasks, and applying operators to primitive tasks to produce actions. If this is done in a way that satisfies all constraints, then the planner has found a solution plan; otherwise the planner will need to backtrack and try other methods and actions. Our version of JSHOP2 applies operators and methods in the order given in the domain description. In the following, primitive tasks are prefixed with ! and variables with ?

3. The course generation planning domain

In this section, we will describe the basic course generation knowledge, that is, a basic set of building blocks from which complex course generation scenarios can be constructed. Before we explain the knowledge in detail, we will briefly discuss the peculiarities of the course generation domain.

A feature that distinguishes the course generation planning domain from other domains is that there exist a number of tasks that should be achieved if possible, but failing does not cause backtracking. We call these tasks *optional* tasks. An example is the motivation of a concept. If there exist educational resources that can serve as a motivation, then they should be included in the course. If no appropriate resource can be found, the course generation should continue anyway, since in this case it is preferable to give the student a sub-optimal course rather than no course at all. In contrast, other tasks are *critical*, and have to be achieved (we will mark these tasks with a ! suffix). Whether a specific task is critical or optional depends on the scenario. Therefore, in PAIGOS, for almost any critical task there exists an equivalent optional task. A second notable feature of the course generation domain is that, in general, resources are not consumed in the sense that they are no longer available. This is different from other domains, such as the travel domain in which fuel is consumed. An educational resource that is added into a course can be added again at a later time (the only potential constraint is that it makes sense from a pedagogical point of view). Similarly, starting/ending a section can be realized at any time and repeatedly without consuming resources. As a consequence, several operators do not have preconditions (neither do some methods) nor delete or add lists.

3.1. Course generation planning problems

The course generator constructs a course for a given user and a learning goal (also called "goal task"). A learning goal is a pair $t = (p,L)$, where *p* represents the pedagogical objective and *L* is a list of educational resource identifiers. *L* specifies the course's target concepts, and *p* influences the structure of the course and the educational resources selected. The order of the resources in *L* is relevant, ie, another order can result in a different course. As an example, the learning goal *t* = (*discover*, (`def_slope`, *def_diff*)) represents the goal of a user who wants to discover and understand the concepts `def_slope` and *def_diff* in depth.

Table 1 contains a selection of goal tasks formalized within PAIGOS, partly designed in cooperation with pedagogical experts.

The general form of a planning problem for a user with identifier *userId* and the learning goal (*ped_Obj* $f_1 \ldots f_n$) looks as follows:

1 *(defproblem Problem CourseGeneration*
2 *(user userId)*
3 *(goalTask (ped_Obj* $f_1 \ldots f_n$*))*
4 *(scenario ped_Obj)*
5 *(targetFundamental* f_1*)* $\cdots$ *(targetFundamental* f_1*)*
6 *((ped_Obj* $f_1 \ldots f_n$*)))*

The first line defines a problem with the name *Problem* to be solved in the domain *CourseGeneration*. The initial state (lines 2–5) contains the user identifier (line 2), the complete goal task (line 3) and the decomposed goal task (split in the scenario name, line 4, and the target resources, line 5). This information is redundant since it can be generated from the goal task, but is nevertheless represented in the initial state to simplify access to that information during course generation. The initial task network consists of the learning goal (line 6). Notably, the world state contains neither information about the resources nor about the learner (besides the identifier). All this information is retrieved dynamically, when required.

3.2. Basic course generation operators and methods

We will now describe the details of the course generation planning domain. The following two axioms implement basic general functionality. The equality of two given terms is tested by the axiom (:- (*same*?*x*?*x*) ()). The axiom's head matches only if the terms passed as arguments are the same. Otherwise the variable *?x* would need to be bound to two different values, which is impossible. If the axiom's head matches, its body is evaluated. Since it is empty, the axiom is satisfied.

In JSHOP2, an assignment expression (*assign*?*var t*) binds *?var* to the term *t*. The following axioms extend this behavior to a list:

(:- (assignIterator ?var (?head. ?tail)) (assign ?var ?head))
(:- (assignIterator ?var (?head. ?tail)) (assignIterator ?var ?tail))

If the precondition of an operator or method contains (*assignIterator ?var termList*), all bindings of *?var* to the elements of the list *termList* will be generated. The first axiom binds *?var* to the first value of the list (split in its head and tail by using the JSHOP2 "·" operator); if the planning process fails at any time later, backtracking causes the second axiom to be applied, which recurses into the list and thus applies the first axiom to bind *?var* to the next value. This process is repeated until the list is empty, which means the axiom cannot be satisfied. Then the planning process backtracks.

The result of a planning process in the course generation domain is a plan consisting of a sequence of operators that, when applied, generates a structured list of references to educational resources and learning-support services. Several operators and methods handle the insertion of references to educational resources in a course. The basic operator is *(:operator (!insertResource ?r) () () ((inserted ?r)))*. It has no precondition and delete list. It adds a logical atom to the world state that describes that a resource *?r* was inserted into the course.

Two methods use this operator. The method *(:method (insertResourceOnce! ?r) ((not (inserted ?r))) ((!insertResource ?r)))* is applicable only if the given resource was not yet inserted into the course and in that case inserts the resource, otherwise it fails. The method shown below inserts a given resource if it was not yet inserted (line 2), otherwise (the second precondition/tail pair, line 3) achieves the task directly since it has no subtasks.

Table 1
A selection of pedagogical objectives used in Paigos.

Identifier	Description
`discover`	Discover and understand fundamentals in depth
`rehearse`	Address weak points
`trainSet`	Increase mastery of a set of fundamentals by training
`guidedTour`	Detailed information, including prerequisites
`trainWithSingleExercice`	Increase mastery using a single exercise
`illustrate`	Improve understanding by a sequence of examples
`illustrateWithSingleExample`	Improve understanding using a single example

```
(:method (insertResource ?r)
((not (inserted ?r))) ((!insertResource ?r))
()())
```

Two additional methods operate over lists of resources. The method *insertAllResources* inserts in a course all resources of a given list. The method *addInWorldStateAsInserted* marks a list of resources as inserted.

The operators *(:operator (!startSection ?type ?refs) () () ())* and *(:operator (!endSection) () () ())* are used for creating structure within a course. The intended meaning is that all references that are inserted between a *!startSection* and an *!endSection* operator are contained within the same section (sections can be nested). The parameters of the *!startSection* operators allow to provide information about the type of the section and the principal resource of the section. This information can be used to generate detailed section titles. For instance, the type *introduction* with reference *def_slope* will generate the section title "Introduction for Definition of a Slope" in several languages. A related operator *(:operator (!text ?type ?parameters))* generates short bridging and introductory texts that explain the purpose of sections.

3.3. Accessing information from external knowledge sources

Planning in the course generation domain requires up-to-date information about the learner (in order to adapt to his competencies and preferences) and about existing resources (for knowing which sections and references can be inserted). In the course generation domain, it is practically impossible to know the world state in advance. Typically, educational resources are too numerous to be represented in the world state. Thus, the planner's information about the world is incomplete. Various approaches in AI address this problem, for instance trough *sensing actions* (Etzioni et al., 1992). Similarly, we propose retrieving the necessary information *during the planning process*. In the following, we will describe the call terms, ie, the external functions that Paigos uses to access information from external knowledge sources.

The function *(GetResources query)* returns the list of identifiers of those educational resources that fulfill the given query. The query consists of a partial metadata description with three parts: the classes of the educational resources, property metadata, and relational metadata. Using a *mediator* architecture, this query is passed to all connected learning object repositories (and translated into their specific terms, if necessary) (Kärger, Ullrich, & Melis, 2006). The set of identifiers that meet the description is then returned to the planner. As an example, the call term *(GetResources ((class Exercise) (relation isFor def_slope)))* returns the list of all exercises for the educational resource with the identifier *def_slope*.

Often, it is necessary to find educational resources which are connected to a given resource by some relation. A typical example is to find the prerequisites of a concept. In Paigos, the function *GetRelated* provides this functionality. Sorting a set of resources with respect to a given relation is done using the function *Sort*.

The function *(learnerProperty ?property ?value ?ref)* accesses information about the learner. It takes a property as input and binds the variable *?value* to the value stored in the learner model for this property for the current user. *?ref* is optional and allows to pass a resource identifier, thereby enabling to query information about the user with respect to a given resource. For example, *(learnerProperty hasEducationalLevel ?edlev)* binds *?edlev* to the educational level of the current learner, e.g., *universityFirstYear* for a first year university student. *(learnerProperty hasCompetencyLevel def_slope ?cl)* binds *?cl* to the current competency level that the learner has reached with respect to the concept *def_slope*, e.g., 3. The specific properties that can be queried depend on the learner model. In ActiveMath, the properties include static information such as the current level of education and his field of interest, and dynamic information such as the competency level, motivational and affective state and whether a resources was already seen. This latter property is used by the axiom *sortByAlreadySeen* to divide a list into unseen and seen resources.

In the PISA competency framework (OECD, 2003) that underlies ActiveMath, a learner who has reached a competency level of three (on a range from 1 to 4) is able to perform extensive computations. In Paigos, this is represented by the axiom *(:- (known ?f) ((learnerProperty hasCompetencyLevel ?f ?cl) ($\geqslant$?cl 3)))*. The axiom is satisfied if the learner has reached a competency level greater or equal to three with respect to the given concept.

Several axioms are based on this elementary axiom: the axiom (*allKnownOrInserted*?*refs*) is satisfiable if all resources in the given list are either known or were inserted into the course.

The axiom *removeKnownOrInsertedFundamentals* removes from a list of concepts[1] those concepts the learner "knows". The axiom *readyAux* checks whether the learner is prepared to understand a given resource by testing whether all concepts that the resource is for are either known or were inserted into the course (in other words whether an opportunity is provided to the learner to understand the necessary concepts before he reaches the resource).

The operator *!insertWithVariantsIfReady* inserts a resource into a course and marks all its variants as inserted. Two educational resources are considered being variants if they are almost equivalent and differ only insignificantly. This fact is represented in the metadata with the relation *is-variant-of*. For instances, two exercises *a* and *b* are marked as being variants if they present the same problem but *a* uses a graph in addition to the text. More often than not, only either *a* or *b* should be included in a course. In paigos, this is achieved by using *!insertWithVariantsIfReady* instead of *!insertResource*.

3.4. Structure and dynamic adaptivity: dynamic tasks

Course generation faces a dilemma: on the one hand it makes sense from a pedagogical point of view to generate a complete course immediately after receiving the learner's request, instead of selecting and presenting one resource after another, as it is done in dynamic content sequencing. The learner is provided with the complete sequence of content that leads him toward his learning goal, and can recognize how the content is structured and can freely navigate.

On the other hand, if a long time passes between the generation and consumption of a page, assumptions about the learner may have become invalid. Then, some presented material would be

[1] The axiom uses the term "fundamental" instead of "concept" due to the terminology used in the ontology used to represent the different types of the educational resources.

inadequate. Hence, the course generation should adapt dynamically and use the most up-to-date information about the learner that is available.

Execution monitoring and re-planning offers a framework that can cope with situations in which assumptions made during planning can change while the plan is executed (for an overview, see Russell & Norvig (2003, p. 441ff)). However, this framework cannot be applied to course generation as realized in PAIGOS. Here, the plan is completely applied before the course is presented, in fact, applying the plan produces the course.

The solution implemented in PAIGOS is based on dynamic subtask expansion. That is, planning may stop at the level of specially marked primitive tasks, called *dynamic tasks*. Each dynamic task encloses a pedagogical task *t* and is inserted into the course instead of *t*. Since a dynamic task is primitive, it counts as directly achieved by its operator and is not further expanded.

Later, at presentation time, when the learner first visits a page that contains the dynamic task, the task *t* it encloses is passed to the course generator. Then, the course generator assembles the sequence of resources that achieve *t*. The resulting identifiers of educational resources replace the dynamic task in the course structure with a list of instances of educational resources. Hence, when the page is revisited, the elements do not change, which avoids confusion of the learner reported by De Bra (2000). This means a course is partly static, partly dynamic, and thus the goal of presenting the complete course to the learner while still being able to adapt is realized.

Dynamic tasks can be introduced by human "course generators", too. That is, Authors who manually compose courses can define a course for which parts of the course are predefined and others dynamically computed. In this way, an author can profit from the best of both worlds: she can compose parts of the course by hand and at the same time benefit from the adaptive features of the course generator.

Dynamic task expansion offers an additional benefit: since the course generation stops at a higher level and does not expand all subtasks, the planning process is much faster. The technical evaluations (see Section 5) show an increase of performance up to a factor of 10 due to this technique.

Since dynamic subtask expansion is not natively supported by JSHOP2's planning algorithm, it is simulated in PAIGOS by the operator *(:operator (!dynamicTask ?ped_Obj ?refs) ()()())*. If a subtask *t* of a method is not to be expanded, then *t* is given as parameter to the primitive task atom *!dynamicTask*. Since the operator that performs this task atom has no preconditions, it can be performed directly. When the operator is applied during plan execution, it creates a special element called *dynamic item*. At a later time, when the course is presented to the learner and the server that handles the presentation detects a dynamic item on the page, it passes the associated dynamic task to the course generator. Then, the course generator assembles the educational resources that achieve the task. In a final step, these resources replace the dynamic item and are presented to the learner.

In order to avoid that new dynamically generated sequences duplicate references to previously inserted resources, dynamic task expansion needs to take the current course into account. Thus, each reference to a resource *r* contained in the current course results in a logical atom *(inserted r)* that is inserted into the world state the course generator is started with.

3.5. Converting a Plan into a Course

After a plan is found, it is used to generate a course, more precisely, a table of contents. PAIGOS represents courses using the element `omgroup`, which is an element from the OMDOC standard, a semantic knowledge representation for mathematical documents (Kohlhase, 2006). The purpose of the `omgroup` element is to represent collections of resources. It is independent of the mathematical domain OMDOC was developed for. It can be easily mapped to other data structures with similar aims, such as IMS CP (IMS, 2003).

An `omgroup` element consist of metadata information (e.g., the author and title of the element), references to other OMDOC elements, other `omgroup` elements, and dynamic items that allow the dynamic inclusion of resources generated by services.

From a plan, a course represented as an `omgroup` is constructed:

- *!startSection* triggers the opening of an `omgroup` element.
- *!endSection* inserts the closing tag of an `omgroup` element.
- *!insertResource* inserts the *ref* element that OMDOC uses for resource references.
- *!text* inserts a *dynamicItem* element that serves as a symbolic representation for text generation.
- *!dynamicTask* inserts a *dynamicItem* element that is used for dynamic task expansion.
- Internal operators (marked with the prefix "*!!*") serve JSHOP2's internal bookkeeping purposes and are ignored.

The resulting OMDOC grouping consists of nested sections in which the leaves are pointers to educational resources.

4. Moderate constructivist competency-based scenarios in PAIGOS

In this section, we explain how we used the pedagogically neutral operators, methods and axioms described in the previous section to build complex course generation scenarios. We will focus on a scenario that should lead to moderate constructivist teaching in which learners play an active role and are to a large extent responsible for the outcome of their learning process. As a result, the course generation scenario aims at supporting the students in structuring their learning activities and developing strategic competence. In addition, this and other scenarios implement a competency-based approach. Competency-based pedagogy argues that literacy in, e.g., mathematics requires mastering different competencies, such as performing calculations and being able to argue mathematically (Niss, 2002).

First this section explains the example and exercise selection, followed by a detailed description of a scenario that supports learners in reaching an in depth understanding of a given list of concepts. Finally, we sketch the other scenarios that we implemented.

4.1. Selecting exercises and examples

The methods presented in this section implement the pedagogical knowledge of selecting exercises that are "appropriate" for the learner. The meaning of "appropriate" differs depending on the individual learner. For instance, if he is highly motivated then a slightly more difficult exercise might be selected. The most relevant factors are the educational level of the learner and his competency level.

The *learning context* of an educational resource describes the educational level of its typical target audience. In general, resources that are selected for a learner should to correspond to his/her educational level. Otherwise, it may be inadequate in its formulation or difficulty, e.g., either too simple or too difficult (think of a second year university student reading a definition for elementary school).

The *competency level* of a resource measures to which extend a specific competency has to be developed by the student in order to

```
 1  (:method (trainWithSingleExercise! ?c)
 2          (;; preconditions
 3           (learnerProperty hasMotivation ?mc ?c)
 4           (≥ ?m 4)
 5           (learnerProperty hasField ?field)
 6           (learnerProperty hasEducationalLevel ?el)
 7           (learnerProperty hasCompetencyLevel ?cl ?c)
 8           (assign ?unsortedExercises
 9                   (GetResources
10                    ((class Exercise)
11                     (relation isFor ?c)
12                     (property hasLearningContext ?el)
13                     (property hasCompetencyLevel (+ 1 ?cl))
14                     (property hasField ?field))))
15           (sortByAlreadySeen ?exercises ?unsortedExercises)
16           (assignIterator ?exercise ?exercises)
17           )
18          (;; subtask
19           (insertWithVariantsIfReady! ?exercise ?c)))
```

Fig. 1. Example of a method for trainWithSingleExercise.

solve/understand the particular exercise/example with a certain probability. In most cases, resources presented to the learner should have a competency level that corresponds to the learner's since these are the resources he is able to understand and work with.

The task that governs the exercise selection is *(trainWithSingleExercise! f)*, which triggers the insertion of an exercise for the concept *f*. All methods that formalize the knowledge of how to select an exercise follow the same basic scheme, which we will explain using the method in Fig. 1. In short, this method specifies that if a learner is highly motivated, then it inserts a subtask that selects an exercise of the next higher competency level. This method is based on the strong positive correlation between motivation and performance (Pintrich, 1999): in general, with increasing motivation, performance increases, too (and vice versa).

In the figure, lines 3–8 prepare the ground for selecting the exercise. The lines 3–4 specify the condition under which the method can be applied (the learner is highly motivated). The axiom *learnerProperty* binds the current motivation represented by a number between 1 and 4 to the variable *?m* (line 3). The expression in line 4 tests whether *?m* is equal to 4, the highest motivational level. The subsequent lines 5–7 collect information about the learner, which is used to specify the metadata constraint, ie, the field of interest of the learner (line 5), his educational level (line 6), and his competency level (line 7). The information collected up to now is used to instantiate a mediator query. The query includes the constraints that the resources have the type *exercise* (line 10) and that they are *for f* (line 11). The competency level that the exercise should be for is one higher than the current competency level of the learner since the exercise should be slightly more difficult (line 13). In case the learner has reached the highest competence level, increasing the value has no effect. In lines 9–14, the query is sent to the mediator. If there exist any educational resources that fulfill the constraint, then these resources are bound to the list variable *?unsortedExercises* in line 8. Line 15 sorts the list and moves any not yet seen resources to the front of the list. The axiom *assignIterator* causes the planner to iterate through the list of exercises (line 16), and the subtask of the method inserts the first exercise that the learner is ready to see (line 19). If there is such an exercise, it is inserted and all its variants are marked as inserted. Otherwise, if none of the exercises bound to the list *?exercises* can be inserted or no exercises was found at all, then the planning algorithm backtracks and applies the next possible operator or method.

All in all, about 60 methods govern exercise and example selection. They are too numerous to describe all of them in detail, and thus we will only describe them briefly in the following. They all follow the scheme explained in Fig. 1, ie, they define a metadata constraint which is send to the mediator.

In case the learner exhibits a low motivation, then an exercise of a lower competence level is presented if available. Otherwise, the course generator tries to insert an exercise whose metadata corresponds directly to the learner's characteristics: if available, an exercise is selected that has the learner's field and corresponds to the learner's educational and competency level. If no exercise was found, then the constraint on the field value is dropped. If still no adequate exercise exists, then several methods search for exercises on the next lower competency level, first with and then without the field constraint. The rationale is that it is better to present a exercise with a too low competency level than one of a different learning context since resources from a different learning context might be harder to understand than "easier" exercises. Finally, the constraint on the learning context is relaxed by including exercises from lower contexts.

An additional method applicable on the task atom *trainWithSingleExerciseRelaxed!* covers least constrained exercise selection. This task serves as a fallback task in case none of the above methods could be applied. The method relies on the belief that presenting a potentially inadequate exercise is preferred over presenting no exercise at all. The need for such a task became apparent in empirical evaluations involving students who preferred inadequate resources over no resources at all. The method omits the constraint on the competency level and traverses all potential educational levels. In addition, it does not check whether the learner is "ready" to understand the exercise but directly inserts it if it was not already inserted (using the task *insertResourceOnce!*).

Sometimes it is necessary to search for exercises that are of a specific competency and difficulty level. Hence, a set of methods equivalent to those described above exists that adds constraints on difficulty and competency. These methods are applicable on the task *(trainWithSingleExercise! f difficulty competency)*.

An additional method applicable on the task *(train! f)* selects a sequence of exercises for a concept *f*. The exercises cover all competencies (as far as there exist adequate exercises for the competencies). A related method for the task *(practiceCompetency competency f)* triggers the insertion of exercises that train a specific competency *competency* for a given concept *f*.

The example selection formalized in Paigos is very similar to the exercise selection and thus will not be covered in detail. The main difference is that the field of an example is considered as being more important than in exercise selection: examples illustrate aspects of a concept and should, if possible, use situations and provide context of the learner's field of interest.

4.2. Scenario discover

The scenario *discover* generates courses that contain those educational resources that support the learner in reaching an in depth understanding of the concepts given in the goal task. The course includes the prerequisites concepts that are unknown to the learner. It also provides the learner with several opportunities to use learning-support services.

The basic structure of the scenario follows the course of action in a classroom as described by Zech (2002), which consists of several stages that typically occur when learning a new concept. For each stage, the course contains a corresponding section. The following sections are created:

Description: The course starts with a description of its aim and structure. Then, for each concept given in the goal task, the following sections are created.
Introduction: This section motivates the usefulness of the concept using adequate resources (for all stages, the precise meaning of an "adequate" educational resources is explained in the formalized methods below). It also contains the prerequisites unknown to the learner.
Develop: This section presents the concept and illustrates how it can be applied.
Proof: For some concepts (theorems), proofs, or more general evidence supporting the concepts are presented.
Practice: This section provides opportunities to train the concept.
Connect: This section illustrates the connections between the current concept and related concepts.
Reflection: Each course closes with a reflection section, which provides the learner with opportunity to reflect on what he has learned in the course.

The two methods illustrated in Fig. 2 start the generation of a course for the scenario *discover*. The upper method decomposes the task (*discover f*) into five subtasks. First, a new section is started, in this case the course itself (line 3). Then, a description about the course's aims and structure is inserted (line 4). The third subtask triggers a method that recursively inserts the task *(learnFundamentalDiscover g)* for each identifier *g* in the list of identifiers bound to *?fundamentals*. The last two subtasks insert the reflection section and close the course.

As a consequence, for each concept *g*, a task *(learnFundamentalDiscover g)* is created. The bottom method in Fig. 2 decomposes the task into subtasks which closely resemble the structure of the scenario as described in the previous section:

An introduction of a concept *f* in the scenario *discover* consists of a section that contains one or several educational resources that introduce *f* and of a section that contains the prerequisite concepts that the learner needs to see. The method is defined as *(:method (introduceWithPrereqSection! ?c) () ((introduceWithSection! ?c) (learnPrerequisitesFundamentalsShortSection! ?c)))*.

```
1  (:method (discover ?fundamentals)
2          ()
3          ((!startSection Discover ?fundamentals)
4           (descriptionScenarioSection ?fundamentals)
5           (learnFundamentalsDiscover ?fundamentals)
6           (reflect ?fundamentals)
7           (!endSection)))
8  (:method (learnFundamentalDiscover ?c)
9          ()
10         ((!startSection Title (?c))
11          (introduceWithPrereqSection ?c)
12          (developFundamental ?c)
13          (proveSection ?c)
14          (practiceSection ?c)
15          (showConnectionsSection ?c)
16          (!endSection)))
```

Fig. 2. Top-level decomposition in the scenario *discover*.

The resources that introduce a concept *f* are determined by the method in Fig. 3. The method starts a new section and inserts a text that explains the section's purpose. The next three tasks try to insert several resources: a resource that motivates the concept, a real-world problem involving *f*, and an example that illustrates the application of *f*. The planning tries to maintain the students motivation by presenting an exercise if there is sufficient evidence that the learner will be able to solve it, and an example or text of type *introduction* otherwise.

In the scenario *discover*, all prerequisite concepts that are unknown to the learner are presented on a single page. Thus, the students easily distinguish between the target concepts and the prerequisites.

The axiom shown in Fig. 4 is used to retrieve the prerequisite concepts. In a first step, all concepts that are required by the concept bound to *?c* and whose learning context corresponds to the educational level of the learner are collected using the function *GetRelated* (lines 3–6). In case some are found (line 7), they are sorted with respect to the prerequisite relationship *requires* (lines 8–13). Finally, those concepts that are known to the learner are removed using the axiom *removeKnownFundamentals* and the result is bound to the variable *?result*.

The axiom is used by the method shown in Fig. 5. It first collects all unknown concepts (in the precondition, line 2) and, if there are any, adds a task that inserts them (line 4).

The section *develop* presents the concept, together with resources that help the learner to understand it. Fig. 6 shows the corresponding method. Both precondition-subtask pairs start a new section, include a text that explains the purpose of the section and insert the concept the section is about. In case the learner

```
(:method (introduceWithSection! ?c)
        ()
        ((!startSection Introduction (?c))
         (text Introduction (?c))
         (motivate! ?c)
         (problem! ?c)
         (insertIntroductionExample! ?c)
         (!endSection)))
```

Fig. 3. *IntroduceWithSection!* generates an introduction.

```
(:- (collectUnknownPrereq ?c ?result)
    ((learnerProperty hasEducationalLevel ?el)
     (assign ?resources (GetRelated (?c)
                          (((class Fundamental)
                            (relation isRequiredBy ?c)
                            (property hasLearningContext ?el)))))
     (not (same ?resources nil))
     (assign ?sorted (Sort ?resources
                         (((class Fundamental)
                           (relation isRequiredBy ?c)
                           (property hasLearningContext ?el)))))
     (removeKnownOrInsertedFundamentals ?reversedUnknown ?sorted)
     (assign ?result (Reverse ?reversedUnknown))))
```

Fig. 4. *CollectUnknowPrerq* collects all unknown prerequisites.

exhibits a high competency level (tested in the first precondition), a single example illustrates the concept. Otherwise, the learner does not exhibit a high competency level, and first a text explaining the concept is inserted, followed by several examples that aim at providing the learner with an understanding of the concept. The example insertion uses the dynamic task *(!dynamicTask illustrate! (?c))*. The planning process does not expand this subtask, hence the specific examples are selected at a later time. The final subtask closes the section. We do not describe the methods for the task *explain* and *prove*. Basically, they search for resources of the respective type.

The methods in the section *practice* insert a list of exercises that provide the learner with opportunities to develop her own understanding of the concept from a variety of perspectives. The method illustrated in Fig. 7 creates a corresponding section. Note that the result of the function *GetResource* (lines 5–9) is not bound to a variable. Its only purpose is to test whether there exists an exercise that can be inserted at a later time, when the dynamic task is expanded. This test is performed for each applicable educational level until matching resources are found. In case no resource was found, the method is not applicable and backtracking takes place. If this test would not be performed, then it might happen that a dynamic task is inserted even if there are no exercises that can instantiate it.

```
(:method (learnPrerequisitesFundamentalsShort! ?c)
        ((collectUnknownPrereq ?c ?result)
         (not (same ?result nil)))
        ((insertAllResources ?result)))
```

Fig. 5. Inserting all unknown prerequisites.

```
(:method (developFundamental ?c)
        ((learnerProperty hasCompetencyLevel ?cl ?c)
         (≥ ?cl 3))
        ((!startSection Title (?c) (developFundamental (?c)))
         (text Develop (?c))
         (!insertResource ?c)
         (illustrateWithSingleExample ?c)
         (!endSection))

        ()
        ((!startSection Title (?c) (developFundamental (?c)))
         (text Develop (?c))
         (!insertResource ?c)
         (explain ?c)
         (!dynamicTask illustrate! (?c))
         (!endSection)))
```

Fig. 6. Developing a concept.

The subtasks of the method start the section and insert a text that explains the purpose of the section (lines 10–11).

Line 12 inserts a reference to a learning-supporting service, called *exercise sequencer*. An exercise sequencer leads the learner interactively through a sequence of exercises until a terminating condition is reached, given by the second parameter. In this case, the parameter *TrainCompetencyLevel* specifies that the learner should reach the next higher competency level. Since some learners prefer not to use the exercise sequencer, the following subtask—a dynamic task—triggers the insertion of exercises (line 13, the task *train!* was explained in Section 4.1). Due to the preconditions, it is certain that this subtask can be fulfilled. The final subtask closes the section.

The final sections, *connect* for each chapter and *reflect* for the overall course aim at meta-cognitive activities. The section *connect* illustrates the connections between the current concept and related concepts such as theorems. The reflection step provides the learner with opportunity to reflect on what he has learned in the course. Preferably, using an Open Learner Model (OLM, Dimitrova, 2002), if such a service is available, or otherwise a text that explains how to reflect.

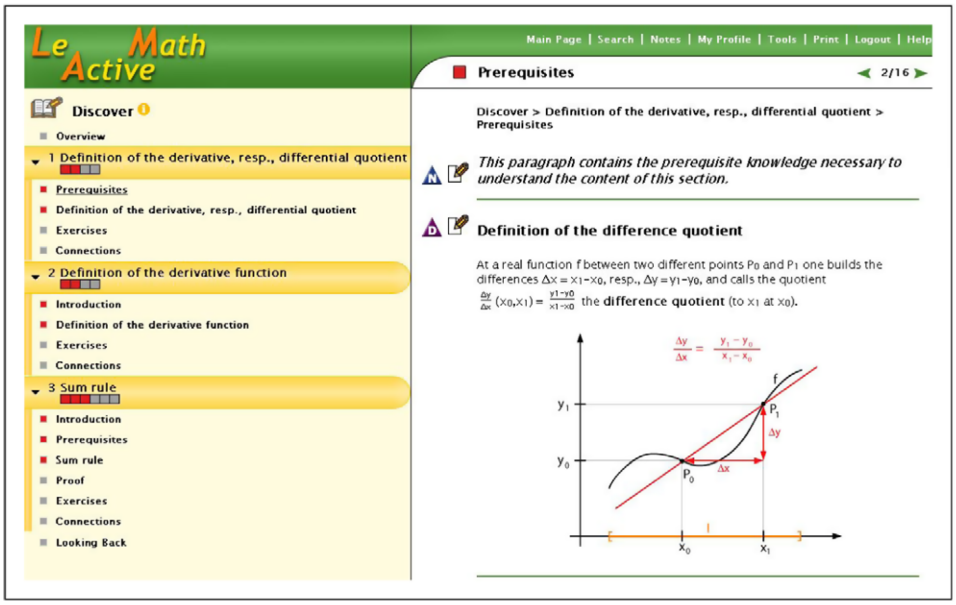
Fig. 8 contains a screenshot of a course generated for the scenario *discover* and the goal concepts "definition of the derivative, resp., differential quotient", "definition of the derivative function" and the theorem "sum rule". The page displayed on the right hand side of the figure is the second page of the course. It contains the first items of the prerequisites page: the generated text that describes the purpose of the section and the first of the prerequisite concepts. The sections displayed in the table of contents vary in the pages they contain. For instance, the first section does not contain an introduction page. The reason is that no elements could be found to be displayed in this section and therefore, the section was skipped.

4.3. Other scenarios

During the learning process, students will have different learning goals. For instance, the scenario *discover* supports students in

```
(:method (practiceSection! ?c)
        ((learnerProperty hasAllowedEducationalLevel ?aels)
         (learnerProperty hasEducationalLevel ?edl)
         (assignIterator ?el (?edl . ?aels))
         (GetResources
          ((class Exercise)
           (relation isFor ?c)
           (property hasLearningContext ?el))))
        ((!startSection Exercises (?c))
         (text Practice (?c))
         (!insertLearningService ExerciseSequencer TrainCompetencyLevel (?c))
         (!dynamicTask train! (?c))
         (!endSection)))
```

Fig. 7. Training a concept.



LeActiveMath
Main Page | Search | Notes | My Profile | Tools | Print | Logout | Help
Prerequisites
2/16
Discover
Overview
1 Definition of the derivative, resp., differential quotient
Prerequisites
Definition of the derivative, resp., differential quotient
Exercises
Connections
2 Definition of the derivative function
Introduction
Definition of the derivative function
Exercises
Connections
3 Sum rule
Introduction
Prerequisites
Sum rule
Proof
Exercises
Connections
Looking Back

Discover > Definition of the derivative, resp., differential quotient > Prerequisites

This paragraph contains the prerequisite knowledge necessary to understand the content of this section.

Definition of the difference quotient

At a real function f between two different points P_0 and P_1 one builds the differences $\Delta x = x_1 - x_0$, resp., $\Delta y = y_1 - y_0$, and calls the quotient $\frac{\Delta y}{\Delta x}(x_0, x_1) = \frac{y_1 - y_0}{x_1 - x_0}$ the difference quotient (to x_1 at x_0).

Fig. 8. A course generated for the scenario *discover*.

learning previously unknown content. In this section, we describe five additional scenarios that help learners at later stages in their learning.

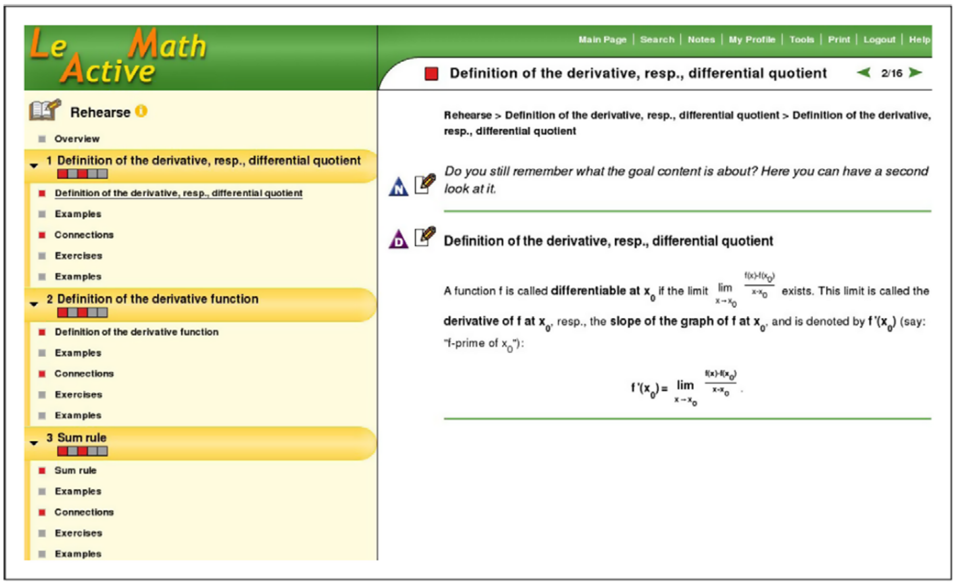
Courses of the type *rehearse* are designed for learners who are already familiar with the target concepts but do not yet master them completely. Such a course provides several opportunities to examine and practice applications of the concepts and illustrates the connections between concepts. The structure is as follows:

Description: The course starts with a description of its aim and structure. Then, for each concept given in the goal task, the following sections are created.
Concept reminder: This section presents the concept of the section.
Illustrate: This section presents example applications of the concept.
Connect: This section illustrate the connections between the current concept and related concepts.
Practice: This section provides opportunities to train the concept.
Illustrate–2: This section contains additional examples.
Practice–2: This section contains additional exercises.

Fig. 9 depicts a screenshot of a course generated for the scenario *rehearse* and the same goal concepts as in Fig. 8. The page displayed on the right hand side of the figure is the second page of the course, which contains the definition rehearsed in the first section of the course. Note that this book is more focused on the selected concepts than the *discover* course (e.g., prerequisites and proofs are omitted) and offers a wider range of exercises and examples.

The scenario *connect* helps the learner to discover connections among the concepts given in the goal task and with other concepts and to provide opportunities to train the concepts. The rationale of this scenario is that laws and theorems connect definitions by describing some relationship between the definitions. For instance, laws in physics put physical concepts in relation to each other. Becoming aware of these connections is beneficial to the user's learning (Novak & Gowin, 1984).

A course generated for the scenario *connect* is structured as follows:


Main Page | Search | Notes | My Profile | Tools | Print | Logout | Help

Definition of the derivative, resp., differential quotient 2/16

Rehearse

Overview

1 Definition of the derivative, resp., differential quotient

Definition of the derivative, resp., differential quotient

Examples

Connections

Exercises

Examples

2 Definition of the derivative function

Definition of the derivative function

Examples

Connections

Exercises

Examples

3 Sum rule

Sum rule

Examples

Connections

Exercises

Examples

Rehearse > Definition of the derivative, resp., differential quotient > Definition of the derivative, resp., differential quotient

Do you still remember what the goal content is about? Here you can have a second look at it.

Definition of the derivative, resp., differential quotient

A function f is called **differentiable at** x_0 if the limit $\lim_{x\to x_0} \frac{f(x)-f(x_0)}{x-x_0}$ exists. This limit is called the **derivative of f at** x_0, resp., the **slope of the graph of f at** x_0, and is denoted by $f'(x_0)$ (say: "f-prime of x_0"):

$$f'(x_0) = \lim_{x\to x_0} \frac{f(x)-f(x_0)}{x-x_0}.$$

Fig. 9. A course generated for the scenario *rehearse*.

Description: The course starts with a section that describes its aim and structure. Then, for each concept given in the goal task, say *A*, the following sections are inserted.
Present concept: The concept is inserted, together with a concept map that shows its connections to laws/theorems and definitions.
Connect: For this section, all definitions (excluding *A*) are retrieved that are required by those theorems that require *A*. Then, for each retrieved definition, say *B*, the following sections are created:
Illustrate: This section presents example applications of the definition.
Train: This section provide opportunities to train the definition.
Develop: This section develops the definition in the following way:
Present: The definition is presented.
Connect-2: This section presents all theorems that connect the original concept *A* with the definition *B* and all previously processed definitions.
Train-Connection: The course ends with a section that contains a concept map exercise.

A course generated for the scenario *train competency* trains a specific competency by presenting sequences of examples and exercises for a specific competency with increasing difficulty and competency level. The courses are structured as follows:

Description: This section provides a description of the scenario. Then, for each concept given in the goal task, the following sections are created:
Rehearse: This section presents the current concept.
Illustrate and practice: The following two sections are repeated for each competency level starting with the competency level one below the learner's current level:
Illustrate: This section contains a sequence of examples for the selected competency and of the selected competency level and of increasing difficulty level.
Practice: This section contains a sequence of exercises for the selected competency and of the selected competency level and of increasing difficulty level.

A course generated for the scenario *train intensively* generates a course that aims at increasing the competency level of the learning by presenting a large selection of exercises. The exercises cover all competencies and are presented in a sequence of increasing difficulty. For each concept, the course consists of two pages containing exercises, the first for the learner's current competency level and the second for the next higher competency level.

The scenario *exam simulation* contains exercises that can be solved within a specified timeframe. In contrast to the previous scenarios, the exercises are not selected with respect to learner properties since this is not the case in a real exam either. The generated courses consists of a number of pages, with each page consisting of exercises for the concepts selected as target fundamentals. The final page of the course contains an estimation of the amount of minutes it takes a average learner to solve all exercises.

An additional scenario, not detailed in this article, is based on Merrill's first principles of instruction Merrill (2002). This scenario served to demonstrate that the pedagogical knowledge formalized in Paigos can realize scenarios following different pedagogical paradigms.

5. Performance of PAIGOS

This section describes the results of a number of evaluations of Paigos that assessed its performance, ie, the time it takes to generate courses under varying conditions. Since Paigos is used for supporting mathematics learning in reality, performance matters. It was not sufficient to develop a prototype, we also had to make sure that the generation time is acceptable to students in real-life usage.

Since the tests were designed to assess the performance of Paigos, influences of other components were minimized as much as possible. The learner model was replaced by a dummy learner model that returned a standard value for each query. In addition, most of the tests were performed with a filled mediator cache. The mediator uses a cache that stores the results of incoming queries. Thereby it avoids sending the same queries repeatedly to the repositories connected to the mediator. For the evaluations, the course was generated once, causing the mediator to store the results of the queries. Then, the actual test runs were performed, on the same set of target concepts and thus resulting in the same queries. In order to minimize influences of latency caused by the network, all components, ie, the testing framework, ActiveMath and its repository and learner model ran on the same computer, a standard PC with 2.8 GH Intel Pentium 4 CPU with 2 GB RAM.

The tests were performed using the scenario *discover*. This scenario includes a large variety of educational resources and is not as specialized as, say, *exam preparation*. The data was collected by generating six courses on 1, 4, 8, 12, 16, and 20 target concepts. Each course generation was repeated 10 times and the data was averaged. Prior to the test, the mediator cache was filled as described above.

Table 2 provides details on the length of the created courses. The course generated for a single concept consists of six pages and a total of 37 educational resources if all dynamic tasks are expanded. A course generated for 20 concepts consists of 83 pages and 365 resources. If each resource had a typical learning time of about two minutes, completing this course would take between 11 and 12 h. These figures illustrate that non-trivial course generation requires a large amount of educational resources: even a course for 4 concepts consists of more than 100 educational resources, such as texts, examples and exercises.

Table 3 and Fig. 10 plot the number of concepts against the time required for course generation (in milliseconds). In the table, the condition **Expanded** provides the time required for a completely expanded course, ie, all dynamic tasks are instantiated. The generation of the smallest course (a single concept, six pages and 37 educational resources in total) takes less than half a second. The generation of a course for 20 concepts takes less than five seconds. The condition **Dynamic Tasks** contains the values obtained for course generation with dynamic tasks. The figures illustrate that using dynamic items rather planning the complete plan can result in a significant performance improvement: a course for 20 concepts is generated in slightly more than half a second. (see Fig. 11).

Table 4 compares the times it takes to generates courses using a filled cache (the same data as in the previous tables) versus an empty cache. The increase is significant: the generation of a course for a single concept with an empty cache takes more than a second, compared to 200 ms with filled cached. A course for 20 concepts takes about 11 s compared to half a second. This data was obtained with the repository running at the same server as the course generator. Hence, accessing a repository over the Web would increase the required time even more.

The reasons for the increase can be found by taking a closer look at the mediator. Table 5 provides details about the number of queries that are sent to the mediator during course generation and about the number of queries that the mediator sends to the repository. The figures differ since the mediator expands a query for a class *c* to include its subclasses. The data is provided for expanded courses (condition **Exp.**) as well as for courses with dynamic tasks (condition **DT**). The high number of queries came as a surprise. Generating an expanded course for a single concept results in about 1 500 mediator queries that are expanded to more than 11000 queries to the repository. The figures are significantly less in condition **DT**. Approximately 150 queries are sent to the mediator, which expands them to about 1200 queries. On the other end of the spectrum, generating an expanded course for 20 concepts results in 21500 mediator queries and more than 100000 expanded queries. Condition **DT** requires approximately 2700 and 14100 mediator and repository queries.

A final test investigated the performance of concurrent access to the course generator. In the test, fifty concurrent course generation

Table 2
The number of concepts, pages and resources of the courses generated in the evaluations.

Number of concepts	1	4	8	12	16	20
Pages	6	19	36	52	79	83
Resources	37	105	202	254	319	365

Table 3
Required time of course generation vs. increasing amount of concepts.

Number of concepts	1	4	8	12	16	20
Expanded	429	1 204	1 875	2 562	3 360	4 834
Dynamic tasks	205	288	415	446	617	617

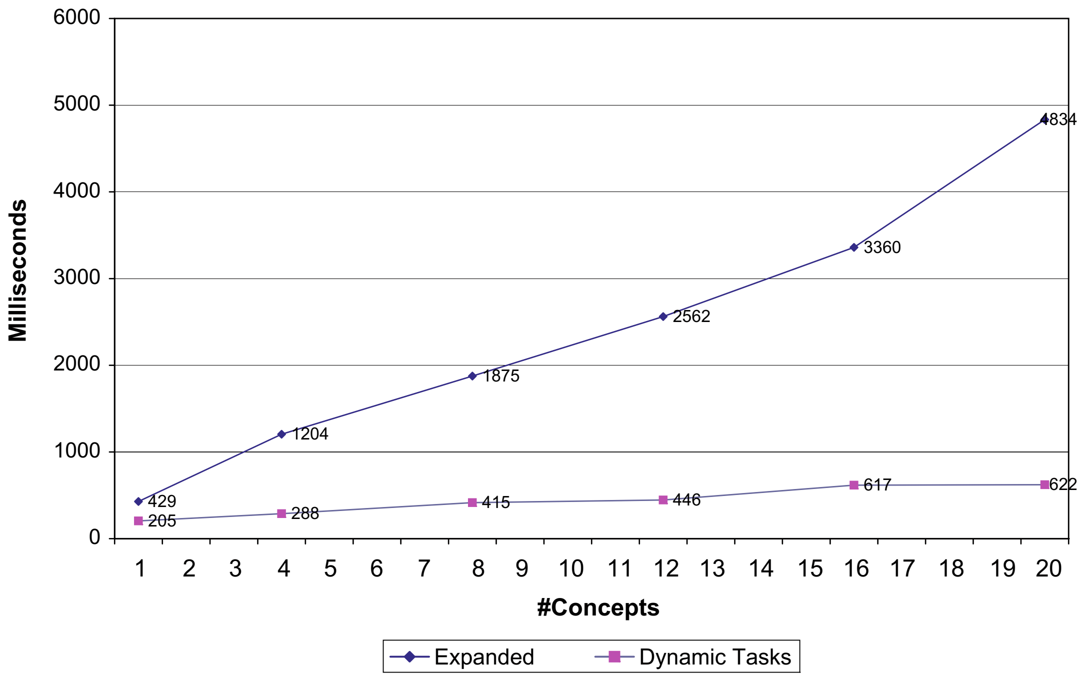


Fig. 10. A plot of the number of concepts vs. time required for course generation in milliseconds.

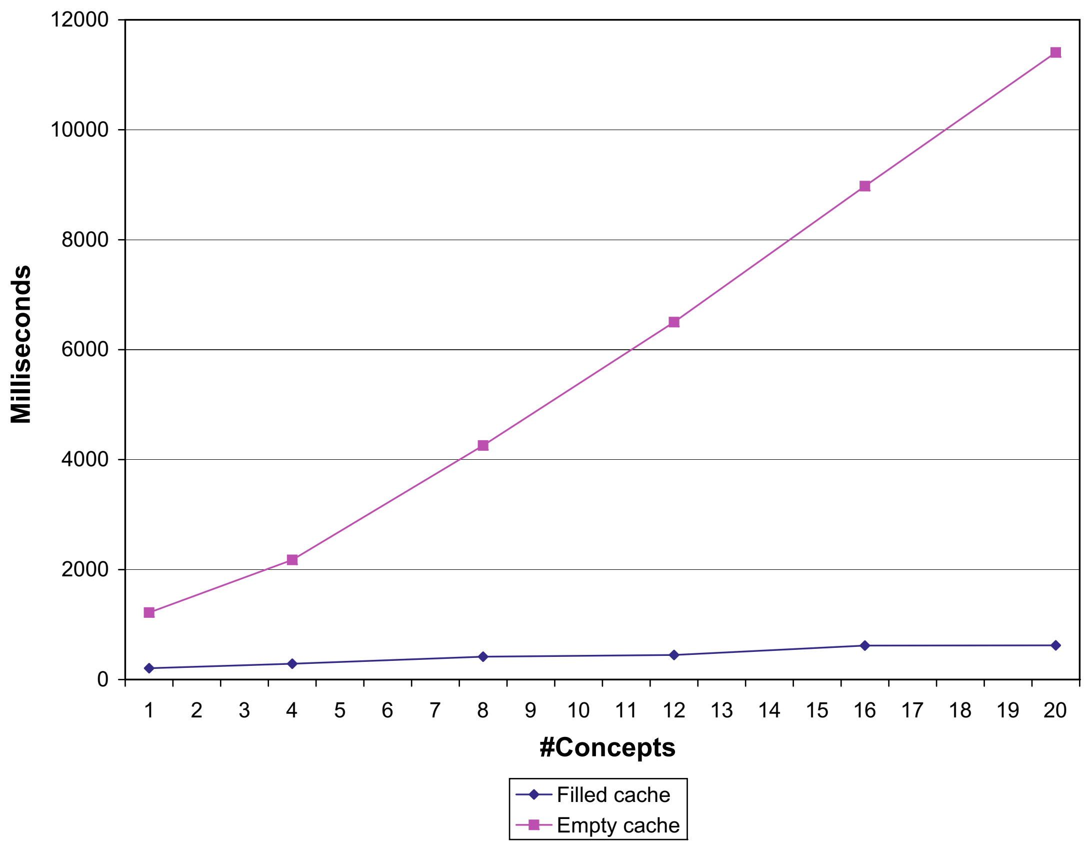


Fig. 11. A plot of the number of concept vs. times required for course generation with empty and filled cache.

processes were started, for a course with four target concepts. Table 6 illustrates the results: on average, a completely expanded course takes 30 s to generate. Using dynamic tasks, it takes about 8 s.

5.1. Discussion

The results show that Paigos on its own generates courses fast, especially if dynamic tasks delay the expansion of subtasks. Thus, dynamic tasks improve the performance and at the same time enable just-in-time adaptivity in generated and authored courses.

The tests were designed to assess the performance of Paigos. As a consequence, external factors needed to be minimized. In particular, Paigos retrieved all resource queries from the mediator's cache. However, this test design is not completely artificial: in classroom usage, each lesson covers a limited set of concepts. After a few course generations, the majority of queries will be cached. In addition, since the topics are known beforehand, the teacher or the e-learning environment can fill the cache before the lesson by letting Paigos generate a course.

Yet, Paigos's real-life performance considerably depends on the repositories and the learner model. In case the components reside on different servers, the network latency alone reduces the overall performance: the LeActiveMath exercise repository is located in Eindhoven, the Netherlands. When accessed from Saarbrücken, Germany, it answers a single query in about 80 ms. As a consequence, the generation of a 4 concepts course that requires 3300 queries requires 4:30 minutes instead of 290 ms.

On the Web, four minutes are an eternity. Few learners will wait patiently for the course to appear: the experiments conducted by

Table 4
Times for generating a course with filled and empty mediator caches.

Number of concepts	1	4	8	12	16	20
Filled cache	205	288	415	446	617	617
Empty cache	1218	2176	4258	6502	8975	11 405

Table 6
Required average time of 50 concurrent course generation processes (in ms).

Dynamic tasks	7 713
Expanded	31 029

Table 5
Numbers of queries to the mediator and of expanded queries.

Number of concepts	1	4	8	12	16	20
Mediator queries (exp.)	1519	5297	9826	13 123	16 901	21 499
Expanded queries (exp.)	11 400	29 697	49 565	62 992	83 923	100 421
Mediator queries (DT)	148	496	1043	1503	2002	2707
Expanded queries (DT)	1204	3349	7810	9702	11 510	14 155

Bhatti, Bouch, and Kuchinsky (2000) showed that subjects rate a response time of over 11 s as unacceptable. This suggests adapting the user interface metaphor of the course generator. Instead of making the users expect an immediate course assembly, a course should be "downloadable", like a PDF document or PowerPoint slides. A download often takes several minutes. After the "download" the user is notified and can access her course from the main menu.

5.2. Formative and summative evaluation

We also performed several formative and summative evaluations in order to assess the students' view on Paigos. The results show that the subjects understand the structure of the generated courses and appreciate the tailoring to their competencies. Those scenarios that support the students in working with exercises are valued highest. Scenarios such as *connect* that are new to students are rated lower.

Open feedback questions used in the evaluations provided other valuable insights. Two subjects commented that they were afraid to miss important content when using generated courses. One of the subject said "the personal book was at a good level, but if I was revising for maths I would be worried that it would miss out something important, and just read the prerecorded books anyway". This underlines the importance of making the concept of course generation familiar to the students.

Another point emphasized by the subjects in the open feedback was that they understand and appreciate the notion that a generated course is indeed personal, that is, it is tailored to and modifiable by the individual learner.

6. Related work

Course generation has long been a research topic. In this section, we will discuss the related work, focused on well-known approaches that use (or claim to use) educational knowledge to inform the sequencing and selection of learning objects.

The idea to encode pedagogical knowledge encoded in planning operators to determine sequences of instructional acts was introduced by Peachy and McCalla (1986). This general idea was refined in the *Generic Tutoring Environment* (GTE) (Van Marcke, 1992) and the *Dynamic Courseware Generator* (DCG) (Vassileva & Deters, 1998). To our knowledge, despite the time that has passed since then, no course generation system except Paigos possesses a comparable large amount of pedagogical knowledge up to now. However, despite the amount of pedagogical knowledge modeled in GTE and DCG, they have no notion of scenarios as developed in Paigos. In addition, the selection of exercises is limited.

DCG introduced a basic technique for course generation that is still frequently used today (e.g., Conlan, Lewis, O'Sullivan, & Wade, 2003). Course generation in DCG is separated in two consecutive steps: the selection of the concepts the course will cover (*content planning*) and the way how these concepts are explained (*presentation planning*).

The distinction between content and presentation planning raises the principal problem that for each concept it is decided separately what educational resources will be selected for it. The selection only takes the current concept into account. Thus, it is not possible to construct a page that contains, say, the definition of a concept and all the definitions of its prerequisite concepts. It is also not possible that the same concept occurs in the course several times, which is necessary, for instance, when a theorem is used in several proofs and should be presented each time. In Paigos, content and presentation planning are interleaved, and this makes the selection much more flexible.

The *Adaptive Courseware Environment* (ACE), developed by Specht and Oppermann (1998) and its successor WINDS (Specht, 2001), offer similar features as DCG and are conceptually very similar to DCG. In particular, they follow the distinction between content and presentation planning and hence suffer from the same drawbacks. In addition, the rules informing the presentation planning are attached to the individual concepts, which is a step backwards from the goal of *generic* instructional knowledge. The same applies to the requirement that the path through the domain structure needs to be specified by the author and is not generated automatically taking into account the learning goals.

Later approaches (Conlan et al., 2003; Keenoy, Levene, & Peterson, 2003; Méndez, Ramírez, & Luna, 2004) put less focus on pedagogical knowledge or do not provide sufficient descriptions to assess it. None of these systems uses the structure of the generated course to convey information about the learning process. In Paigos, the hierarchical knowledge encoded in the pedagogical methods is reflected in the different sections and sub-sections, which provide a means for the student to understand the structure of a course.

Karampiperis and Sampson (2005) suggest to use statistical methods for determining the best path through the educational resources. A major drawback is that the educational resources are rated by an instructional designer with respect to a specific learner. The rating does not take into account that the same learner can have different learning goals regarding the same concepts.

Sicilia, Sánchez-Alonso, and García-Barriocanal (2006) present the idea to use HTN-planning to generate IMS LD instances. Their description is on a very abstract level and sketches some basic operators and methods. By using two HTN methods as an example, they hint at how different pedagogical approaches ("content-oriented" vs. "socio-cultural") might be implemented. Their work provides evidence that course generation techniques can be used for the generation of learning designs, however they do not provide any detailed formalization as done within the context of this thesis.

7. Conclusion

In this article, we described Paigos, a course generator able to generate courses adapted to different learning goals based on an explicit representation of pedagogical knowledge.

In Paigos, the pedagogical knowledge used for course generation is specified as basic building blocks. From these, seven different scenarios were formalized based on learning theories, such as moderate constructivism and based on guidelines from instructional design, resulting in about 300 methods. This illustrates the flexibility and pedagogical neutrality of Paigos. The formalized scenarios generate courses for a range of learning goals such as discovering new content, rehearsing and preparing for an exam. The pedagogical knowledge allows to retrieve very specific content such as adequate examples and exercises. These granular tasks combined with dynamic subtask expansion yield a combination of manually assembled courses with an adaptive selection of resources. They also allow other learning-support services to delegate specific functionality to Paigos. For instance, Paigos can be used for pedagogically founded exercise selection. In ActiveMath, several components use this functionality, such as a suggestion component and an assembly tool.

The generated courses are structured according to pedagogical principles and contain bridging texts that describe the purpose of the sections. Together, these two features provide the student with information that may help him to understand why a course is structured in a specific way.

The architecture and the pedagogical knowledge were implemented in the HTN-planner JSHOP2. Paigos is completely integrated in ActiveMath and is used by several of its components.

The technical evaluation shows that Paigos is efficient. Due to the hierarchical planning, courses consisting of more than 300 educational resources are generated in less than a second.

In future research we would like to explore how to extend Paigos with Web 2.0 techniques and frameworks. Currently, educational resources that are used in Paigos are stored in content sources that have to be registered at the mediator. However, today a vast amount of information is syndicated using standards such as RSS and Atom and annotated with tags. Integrating such content in a generated course might offer a way to overcome the bottleneck of the required large numbers of educational resources.

Acknowledgement

This publication partially is a result of work in the context of the LeActiveMath funded under the 6th Framework Program of the European Union (Contract IST-2003-507826). The authors are solely responsible for its content.

References

Bhatti, N., Bouch, A., & Kuchinsky, A. (2000). Integrating user-perceived quality into web server design. In *Proceedings of the 9th international World Wide Web conference on computer networks: the International Journal of Computer and Telecommunications Netowrking* (pp. 1–16). Amsterdam, The Netherlands: North-Holland Publishing Co. http://dx.doi.org/10.1016/S1389-1286(00)00087-6.

Brusilovsky, P., & Millán, E. (2007). User models for adaptive hypermedia and adaptive educational systems. In P. Brusilovsky, A. Kobsa & W. Nejdl (Eds.), *The adaptive web. Lecture Notes in Computer Science* (Vol. 4321, pp. 3–53). Springer.

Brusilovsky, P., & Vassileva, J. (2003). Course sequencing techniques for large-scale webbased education. *International Journal of Continuing Engineering Education and Lifelong Learning, 13*(1/2), 75–94.

Conlan, O., Lewis, D. Higel, S., O'Sullivan, D., & Wade, V. (2003). Applying adaptive hypermedia techniques to semantic web service composition. In P. de Bra (Ed.), *Proceedings of AH2003: Workshop on adaptive hypermedia and adaptive web-based systems, Budapest, Hungary, May 20–24* (pp. 53–62).

De Bra, P. (2000). Pros and cons of adaptive hypermedia in web-based education. *Journal on CyberPsychology and Behavior, 3*(1), 71–77.

Dimitrova, V. (2002). STyLE-OLM: Interactive open learner modelling. *International Journal of Artificial Intelligence in Education, 13*, 35–78.

Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., & Williamson, M. (1992). An approach to planning with incomplete information. In B. Nebel, C. Rich, & W. R. Swartout (Eds.), *Proceedings of the third international conference on principles of knowledge representation and reasoning, Cambridge, MA, USA* (pp. 115–125). San Mateo, CA, USA: Morgan Kaufmann publishers Inc. ISBN: 1-55860-262-3. <citeseer.ist.psu.edu/etzioni92approach.html>.

Ilghami, O., & Nau, D. S. (2003). A general approach to synthesize problem-specific planners. Technical Report CS-TR-4597, Department of Computer Science, University of Maryland.

IMS global learning consortium. IMS content packaging information model, June 2003.

Karampiperis, P., & Sampson, D. (2005). Automatic learning object selection and sequencing in web-based intelligent learning systems. In Ma Zongmin (Ed.). *Web-based intelligent e-learning systems: technologies and applications* (pp. 56–71). Information Science Publishing (chapter III).

Kärger, P., Ullrich, C., & Melis, E. (2006). Integrating learning object repositories using a mediator architecture. In W. Nejdl & K. Tochtermann (Eds.), *Innovative approaches for learning and knowledge sharing. Proceedings of the first European conference on technology enhanced learning, Heraklion, Greece* (Vol. 4227, pp. 185–197). Springer-Verlag. ISBN: 9783540457770. <http://www.carstenullrich.net/pubs/kaergeretal-mediator-ectel06.pdf>.

Keenoy, K., Levene, M., & Peterson, D. (2003). Personalisation and trails in self e-learning networks. WP4 Deliverable 4.2, IST Self E-Learning Networks.

Kohlhase, Michael (2006). *OMDoc – An open markup format for mathematical documents*. Springer-Verlag. ISBN: 3540378979.

Mayes, J. T. (1992). Cognitive tools: A suitable case for learning. In P. A. M. Kommers, D. H. Jonassen, & J. T. Mayes (Eds.), *Cognitive Tools for Learning, volume 81 of NATO ASI Series, Series F: Computer and Systems Science* (pp. 7–18). Berlin, Heriot-Watt University, UK: Springer-Verlag.

Melis, E., Goguadze, G., Homik, M., Libbrecht, P., Ullrich, C., & Winterstein, S. (2006). Semantic-aware components and services of ActiveMath. *British Journal of Educational Technology, 37*(3), 405–423.

Méndez, N. D. D., Ramírez, C. J., & Luna, J. A. G. (2004). IA planning for automatic generation of customized virtual courses. In *Frontiers In Artificial Intelligence And Applications. Proceedings of ECAI 2004* (Vol. 117, pp. 138–147). Valencia, Spain: IOS Press.

Merrill, M. D. (2002). First principles of instruction. *Educational Technology Research & Development, 50*(3), 43–59.

Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Munoz-Avila, H., Murdock, J. W., et al. (2005). Applications of shop and shop2. *IEEE Intelligent Systems, 20*(2), 34–41. ISSN: 1541-1672. http://dx.doi.org/10.1109/MIS.2005.20.

Nau, D. S., Muñoz-Avila, H., Cao, Y., Lotem, A., & Mitchell, S. (2001). Total-order planning with partially ordered subtasks. In B. Nebel (Ed.), *Proceedings of the 17th international joint conference on artificial intelligence, IJCAI 2001* (pp. 425–430). Seattle, Washington, USA: Morgan Kaufmann.

Niss, M. (2002). Mathematical competencies and the learning of mathematics: the danish KOM project. Technical report, IMFUFA, Roskilde University.

Novak, J. D., & Gowin, D. B. (1984). *Learning how to learn*. New York: Cambridge University Press.

OECD (Ed.) (2004). Learning for tomorrows world — First results from PISA 2003. Organisation for Economic Co-operation and Development (OECD) Publishing.

Peachy, D. R., & McCalla, G. I. (1986). Using planning techniques in intelligent tutoring systems. *International Journal of Man–Machine Studies, 24*(1), 77–98.

Pintrich, Paul R. (1999). The role of motivation in promoting and sustaining self-regulated learning. *International Journal of Educational Research, 31*, 459–470.

Prenzel, M., Carstensen, C. H., & Ramm, G. (2004). PISA 2003 – Eine Einführung. In PISA-Konsortium Deutschland (Ed.), *PISA 2003 – Der Bildungsstand der Jugendlichen in Deutschland – Ergebnisse des zweiten internationalen Vergleichs* (pp. 13–46). Münster, Germany: Waxmann Verlag.

Russell, S. J. & Norvig, P. (2003). *Artificial intelligence: A modern approach*. Pearson Education. ISBN: 137903952.

Sacerdoti, E. (1975). The nonlinear nature of plans. In *The proceedings of the fourth international joint conference on artificial intelligence, Tiblisi, USSR* (pp. 206–214). Morgan Kaufmann.

Sicilia, M.-A., Sánchez-Alonso, S., & García-Barriocanal, E. (2006). On supporting the process of learning design through planners. In *Virtual campus 2006 post-proceedings. Selected and extended papers* (pp. 81–89).

Specht, M., Kravcik, M., Pesin, L., & Klemke, R. (2001). Authoring adaptive educational hypermedia in WINDS. In N. Henze (Ed.), *Proceedings of the ABIS 2001 workshop*.

Specht, M., & Oppermann, R. (1998). ACE – adaptive courseware environment. *The New Review of Hypermedia and Multimedia, 4*, 141–162.

Tate, A. (1977). Generating project networks. In *Proceedings of the fifth international joint conference on artificial intelligence* (pp. 888–893). Morgan Kaufmann.

Ullrich, Carsten (2005). The learning-resource-type is dead, long live the learning-resource-type! *Learning Objects and Learning Designs, 1*(1), 7–15<http://www.carstenullrich.net/pubs/Ullrich-LearningResource-LOLD-2005.pdf> .

Van Marcke, K. (1992). Instructional expertise. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Proceedings of the second international conference on intelligent tutoring systems, Montréal, Canada. Lecture Notes in Computer Science* (Vol. 608, pp. 234–243). Heidelberg: Springer. ISBN: 3-540-55606-0.

Vassileva, Julita, & Deters, Ralph (1998). Dynamic courseware generation on the www. *British Journal of Educational Technology, 29*(1), 5–14.

Wiley, D. A. 2000. Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. In D. A. Wiley (Ed.), *The instructional use of learning objects: Online version*. published online.

Zech, F. (2002). *Grundkurs mathematikdidaktik*. Weinheim: Beltz Verlag.