User-defined Learning Strategies

Wolfgang Theilmann[1], Wolfgang Gerteis[1], Jana Abbing[2]

[1]SAP Research, CEC Belfast
University of Ulster, TEIC Building
Newtownabbey BT37 0QB
United Kingdom
wolfgang.theilmann@sap.com
wolfgang.gerteis@sap.com

[2]TU Darmstadt - FB 20
FG Telekooperation
Hochschulstraße 10
D-64289 Darmstadt
Germany
jana@tk.informatik.tu-darmstadt.de

Abstract: In contrast to traditional learning material, e-learning promises the benefit to support the adaptation of its content according to learners' needs. Learning strategies are used to assure that those adaptations still make pedagogically sense. However, traditional learning strategies are realized at programming level and thus cannot be easily created or changed by pedagogic experts.

This paper presents (1) a general formalism to consistently describe a large range of learning strategies separated from any actual content, (2) an algorithm that applies a learning strategy to arbitrary content in an optimal manner and (3) a graphical notation that allows pedagogical experts to create new or change existing strategies without any programming experience.

1 Introduction & Related Work

One of the often claimed major advantages of e-learning in contrast to traditional learning methods is that the content can be specifically adapted for each learner according to the learner's knowledge and his preferred learning style. Instructional strategies are the basis for guiding a learner through a course in such a way that both the individual learning situation and the instructional concerns are satisfied [Th02].

However, the actual support for learning strategies is still highly restricted. Traditional systems (e.g. [Sp02]) simply rely on some templates i.e. course skeletons built with some pedagogical rationale. More advanced support for adaptation is provided by approaches like [ADL04]. Herein the so-called „Simple Sequencing" provides rich means for steering the sequencing through the course. However, the used primitives are highly technical, hard to understand for pedagogic experts and must be tuned for each course separately. The L^3 project [GA04] came up with the idea of strictly separating learning strategies from actual content and content structures. Within L^3 a course can be sequenced under any of the predefined strategies. However, strategies still need to be programmed by a technical expert and course authors have no mean for their adaptation.

More recent work concentrates on providing more flexibility of the pedagogical models on different levels (e.g. EASE [Ar04], InCA [Na05] and Learning Design Palette [IM04]). One of the most sophisticated approaches and complex support for the author is provided in Adaptive Course Construction Toolkit [Da05]. Author can choose from predefined pedagogical templates (Narrative Concepts, e.g. Case Based Learning), this can be adjusted and linked with concrete learning materials. Finally the author can choose Adaptive Axes for selected parts of the course, which will be then personalized for each learner (according to e.g., prior knowledge, learning styles and context).

Adaptive Hypermedia is an alternative to the traditional „one-size-fits-all“ approach in the development of hypermedia systems. Adaptive hypermedia systems build a model of the goals, preferences and knowledge of each individual user, and use this model throughout the interaction with the user, in order to adapt to the needs of that user [Br01]. Adaptive web-based educational systems (AWBES) are one of the most researched areas within Adaptive Hypermedia. Yet, just a handful of these systems are actually being used for teaching real courses, typically in a class lead by one of the authors of the adaptive system [Br04]. A study [Mo05] among the best practice teachers in UK pointed out that traditional AWEBS lack the support for different pedagogical models and the possibility for personalization according to different groups of students. Interviewed teachers are calling not only for support of predefined learning styles, but also to be able to influence the differentiation by a course-author.

The common challenge to all these approaches is about providing sound pedagogical background along with flexibility for the course authors without increasing the complexity of authoring process.

This paper follows the idea to separate a strategy model from a content model. We take this idea further as we support both, the content and the strategy development in a graphical environment without any need for programming skills. Content and strategies can be automatically transformed into one uniform technical representation which allows one single algorithm to support the application of arbitrary strategies to arbitrary content. This approach combines some unique advantages: It allows clearly separating the roles of a strategy designer and a course author while still allowing course authors to adapt learning strategies according to their needs. No technical/programming skills are required to implement a new strategy. Overall, the approach significantly reduces the complexity of designing courses according to sound pedagogical rationale while still allowing for highly flexible adaptations. The remainder of this paper is structured as follows: Section 2 sketches the overall process and some basic definitions. Section 3 describes the formal Representation of strategies and content. The application of strategies is presented in Sections 4 and 5 while Section 6 discusses the graphical notation of strategies. A summary in Section 7 concludes the paper.

2 General approach

The main idea that we follow is the concept of strategy templates that define learning strategies in a generic way. In order to allow for generic algorithmic treatment of learning units and strategy templates, we introduce formal representations for both of them. These representations are used by a generic mechanism that is able to assess different navigation options within a learning unit according to a given strategy template. This assessment is then used as basis for an overall navigation support algorithm.

Both, strategy templates and learning units are specified in a graphical environment. These representations are then mapped to formal, textual representations, which can be easily used by algorithms. Based on the formal representations, we introduce a generic strategy assessment algorithm, which assesses navigation options within a learning unit according to a given strategy. Finally, the assessments are used for providing overall navigation support to the user.

The following picture gives an overview of the general approach:

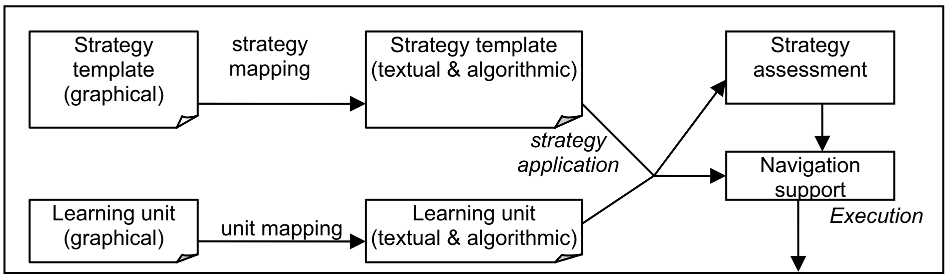
Strategy template (graphical)
strategy mapping
Strategy template (textual & algorithmic)
strategy application
Strategy assessment
Navigation support
Execution
Learning unit (graphical)
unit mapping
Learning unit (textual & algorithmic)

Figure 1: Approach for applying learning strategies.

Inspired by the didactic models proposed by [Me99] and e.g. implemented in [GA04] courses and strategies are constructed in the following way.

Structural elements of courses. We assume an e-learning course to consist of several learning units each of which covers a certain topic in a comprehensive and self-contained way. Learning units consist of knowledge items that have an assigned knowledge type [Me99] and possibly other meta data such as media type.

In addition, there might be some relations between knowledge items that explain the matter-of-fact relationship between these items. We distinguish 2 types of matter-of-fact relationships (that are very similar to the relation types used within strategy descriptions): “before” and “belongs to”. Two concrete knowledge items can be related by a “before” relationship in order to enforce a special order for the sequencing of the items. For example, an author might use the “before” relation in order to relate 2 knowledge items that carry explanation knowledge where the second item builds upon the first one. The “belongs to” relation describes that 2 or more knowledge items belong to each other due to their matter of facts nature. For example, an author designing a learning unit with 2 pairs of example and explanation items might use this relation in order to relate each example with its corresponding explanation.

Learning strategies serve for defining rules for sequencing the knowledge items within a learning unit when the unit is presented to a learner. A strategy template can be described by a set of abstract knowledge items described by a knowledge type and possibly other meta data. Abstract knowledge items can be related to each other. We distinguish 2 relation types: "didactically before" and "associated with". The semantics of the relation types are defined as follows: "didactically before" gives a recommendation on the order of knowledge items how they should be sequenced for the learner. "associated with" describes that several knowledge items are closely related so that they should be presented as a dedicated sub part of the sequencing process. However, it does not specify any policy on the order among the associated items.

Running example. The following simple example serves to illustrate our approach. We assume knowledge items to be attributed with one of the following knowledge types: overview, action, explanation or reference. The example covers a learning unit with 5 knowledge items linked together by a 2 relation types and a strategy template for supporting explanation-oriented knowledge transfer with 3 template knowledge items.

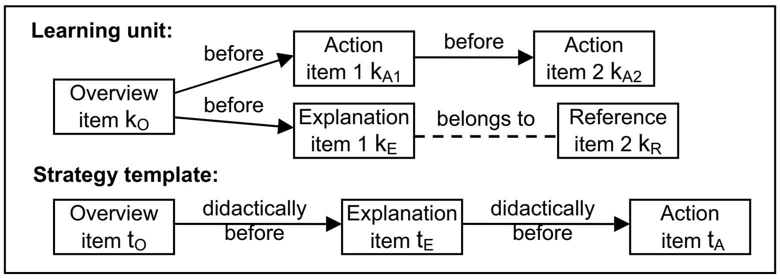

Figure 2. A graphical representation (compare with Section 6)

3 Representation of strategy templates and learning units

Conditional regular expressions. As a textual representation for both strategy templates and learning units we use conditional regular expressions which are defined as follows: Regular expressions are a well known formalism that is based on a set of symbols S [Ma89]. Conditional regular expression are defined like ordinary regular expressions where each (part of a) regular expression r can be guarded by a conditional c which itself is a boolean expression. A regular expression with condition is denoted as c:r.

We further introduce a "visited" attribute v which is actually evaluated while accepting a word that is specified by a regular expression. v becomes true for a part of a regular expression as soon as the part of the regular expression has been accepted by the corresponding automaton and is false before.

We introduce the abbreviations $<r_1,\ldots,r_n>$ (to denote a permutation of all regular expressions $r_1,\ldots,r_n$), $\{r_1,\ldots,r_n\}^*$ (to denote a subset of the regular expressions $r_1,\ldots,r_n$) and $\{ r_1,\ldots,r_n\}+$ (to denote a non-empty subset of the regular expressions $r_1,\ldots,r_n$).

Corresponding to the construction of finite automatons for ordinary regular expressions, we construct finite automatons for our conditional regular expressions. The only difference to an ordinary one is that transitions may have conditions which evaluate to true or false during the acceptance. We denote automatons as follows: Let $A = (Q, S, P, i, F)$ be a deterministic finite automaton where $Q = \{q_1,\ldots,q_n)$ is the set of states of the automaton, $S = \{s_1,\ldots,s_n\}$ is its alphabet, P is a set of transitions $qs::=q'$ with $q, q' \in Q$, $s \in S$; $i \in Q$ is the start state of the automaton and $F \subset Q$ a set of final states of the automaton.

3.1 Textual representation of strategy templates and learning units

Any strategy template can be formally represented as a conditional regular expression. The finite set of symbols S consists of a set of abstract knowledge items that are of interest for the specific strategy. Each abstract knowledge item has some properties (e.g. a knowledge type) that make it different from all other abstract knowledge items within S. Relations between abstract knowledge items are expressed by the structure of the regular expression.

Any learning unit can be formally represented as a conditional regular expression. The finite set of symbols S is simply the set of knowledge items used within the learning unit. Relations between knowledge items are expressed by the structure of the regular expression.

Running example. Preceding the transformation of graphical notations into an associated regular expressions, learning unit and strategy template of the running example are expressed as $\{k_O,$ visited(k_O): $(k_{A1},$ visited(k_{A1}): $k_{A2})$, visited(k_O): $<k_E, k_R>\}^*$ respectively $<t_O,$ visited(t_O):t_E, visited(t_E):$t_A>$.

3.2 Algorithmic representation of strategy templates and learning units

As both, strategy templates and learning units, can be described by conditional regular expression, it is also possible to represent them as deterministic finite automatons [Ma89]. Such a representation is especially useful for algorithms that compute possible sequences and will be heavily used in later sections.

Each transition in such an automaton represents a symbol of the regular expression which in turn represents a knowledge item within the strategy template/learning unit or a condition given by the structure of the regular expression.

4 Strategy assessment

When applying an abstract knowledge item sequence defined by a strategy template to a concrete sequence of knowledge items given by a learning unit this may lead to contradictions regarding the sequence. In order to resolve these conflicts we need to somehow assess the quality of a sequence given by a learning unit according to the one predefined by the strategy template. Throughout the following sections we use the term *strategy item* when talking about knowledge items of a strategy template whereas we say *learning unit item* for a knowledge items contained in a learning unit.

Correspondence. For comparing knowledge items from learning unit with knowledge items from strategy template we a assume an arbitrary correspondence predicate ≈. A simple concrete definition of the corresponding predicate might follow the definition of the knowledge type hierarchy. According to this hierarchy, a learning unit item e_L corresponds to a strategy item e_S if and only if the knowledge type of e_L is of the same type or of a subtype of the knowledge type of e_S

Quality measure. The following distance measure is used to assess the quality of an actual learning unit sequence with respect to an abstract sequence provided by a strategy.

Let seq_S be a sequence according to a strategy and seq_L be a sequence of an actual learning unit. Then, the distance between these sequences is defined as

$$\Delta(seq_S, seq_L) := \text{weighting}(\text{\# missed strategy items, \# superfluous learning unit items, deviation of learning unit item order})$$

where (based on the correspondence predicate ≈)

- # missing strategy items := seqS – seqL, i.e. the items contained in seqS but not in seqL
- # superfluous learning unit items := seqL - seqS, i.e. the items contained in seqL but not in seqS
- deviation of learning unit item order := minimum number of knowledge item exchange operations in seqS $\cap$ seqL to turn it compliant with the order in seqS
- the weighting function can be arbitrarily defined.

4.1 Global strategy assessment

Both, a learning unit and a learning strategy typically do not specify a single sequence but a set of possible sequences. For computing an optimal learning unit sequence (with respect to a strategy) we just need to compute the above introduced distance measure for any pair of possible sequences and select the one with the minimal distance.

However, as learning units and strategies are represented as regular expressions the respective sequence sets may be infinite. A simple heuristic to cope with this in practice is to limit the number of allowed repetitions (denoted by an asterix within the regular expressions) by the maximum number of knowledge items occurring either in the learning unit or the strategy expression. However, even this heuristic can be computationally quite expensive as it may require a processing time exponential to the number of knowledge items

4.2 Local strategy assessment

This section presents a heuristic for assessing different sequencing options within a learning unit according to a given strategy template. It performs a local assessment that can be applied at any stage of the navigation through a learning unit. The assessment is based on a heuristic which is much more efficient than the global optimization presented above. The local strategy assessment will set the base for an overall navigation algorithm, presented in the subsequent section.

The algorithm for assessing options during the execution of a learning unit is based on a finite automaton for each the strategy template and the learning unit. Having given a specific state in the learning unit automaton it evaluates a set of potential strategy items. For each item, an assessment according to the above introduced quality criteria is made. The selected item finally drives the execution of both automatons. The assessment uses the following input parameters:

- q_L := current state of learning unit automaton
- q_S := current state of strategy template automaton
- notVisitedStrategyItems := set of all strategy items that have not been visited yet

```
assessment(q_L, q_S, notVisitedStrategyItems) {
  // compute next reachable transitions in learning unit automaton
  // that correspond with a strategy item
  candidateTransitions := computeTransitionCandidates(q_L);

  // assess candidates
  for each (q*_L, e*_L) ∊ candidateTransitions) {
    // assess missed strategy items by first computing the not yet visited
    // but still reachable strategy items in the learning unit automaton
    reachableItems := computeReachableStrategyItems(q*_L,e*_L);
    numberOfMissedItems := |notVisitedStrategyItems - reachableItems|;

    // assess superfluous learning unit items
    numberOfSuperfluousItems := distance(q_L,e*_L);
    // assess strategy target and deviation of learning unit item order
    (strategyCandidate[q*_L,e*_L], deviation)
        := computeTargetAndDeviation(q_S,e*_L);

    // compute weighted assessment value
    assessment[q*_L,e*_L] := weightingFunction(numberOfMissedItems,
                               numberOfSuperfluousItems, deviation);
  }
  // return candidates and assessments
  return (candidateTransitions, assessment[candidateTransitions],
         strategyCandidate[candidateTransitions]);
}
```

Due to space limitations, we just briefly sketch the semantics the various subroutines.

computeTransitionCandidates(q_L): For a given state q_L of the learning unit automaton the set of transition candidates contains the next reachable transitions in the learning unit automaton that correspond (≈) with a knowledge item of the strategy automaton.

`computeReachableStrategyItems(q*`$_L$`,e*`$_L$`):` For a given transition this set contains all still reachable, but not yet visited strategy items each of which has the following characteristic: starting from the target state of the given transition a learning unit item is reachable that corresponds to a not yet visited strategy item. In the assessment algorithm we derive from this set those strategy items that have not been visited and will definitely not be reachable when the transition candidate is applied. This is the heuristic that we use to estimate the number of the finally missed strategy items for a specific transition candidate.

`distance(q,e):` This is the minimum length of a direct path from q to e.

`computeTargetAndDeviation(q`$_S$`,e*`$_L$`):` Computes the closest reachable transition in the strategy automaton starting from a state q_S that corresponds with the given knowledge item and the deviation for this transition from q_S. It follows the strategy automaton starting from a state q_S and computes those transitions in the strategy automaton that can be reached from that state and that (more precisely: the associated knowledge item) correspond with the given knowledge item of the unit automaton. It selects that transition that has the minimal distance from the given strategy automaton state. Finally, it returns the selected transition and the associated deviation. If no corresponding transition exists, nil is returned as the target and the number of states of the strategy automaton as deviation

The following aspects of the presented algorithms are worth mentioning:

- The weighting function can be arbitrarily specified. Thus, different heuristics can be actually exploited by the algorithm.

- The assessment of the deviation of learning unit item order does not strictly correspond with the quality measure introduced before. This is because the measure works on complete sequences and thus cannot be broken down to local situations. However, the used local assessment is by its nature pretty similar to the global measure.

- The computational complexity of the presented algorithm is linear in the number of states and transitions of the respective automatons. The same applies to any shortest path computation. Thus, the overall computational complexity is O(|states in unit automaton|*max(|states and transitions in unit automaton|, |states and transitions in strategy automaton|).

- The actual evaluation of conditions depends on the nature of the respective condition. Some conditions (for example visited attributes) can be evaluated depending on the previous sequencing through the respective automaton. Others (for example test results) need to be evaluated by some heuristics. E.g. for tests, we may assume that they are always passed.

5 Overall navigation support

The overall navigation support is done in the following way. At any point in time the user gets the option of being guided by the system by just clicking on a "Next" button. In addition, he might get several navigation options presented among which he has to make a choice (same approach as in [GA04]).

The navigation through a learning unit is always accompanied by the traversal of the associated learning unit and strategy template automatons. Thus, at any point in time, both automatons are within one specific state. Based on the current state of both automatons and a specified weighting function (or even a set of specified weighting functions) the system can compute and assess the various navigation options. The best of all options will be associated with the guided navigation (Next button). Based on an arbitrary heuristic (e.g. best 5 or best of each weighting function) the system may also present several navigation options for the user's choice.

Once, a user has chosen a specific option, the system knows about its associated target state (in both automatons). It will then forward the learning unit automaton one step towards the specified target state. If (by this step) the target step is reached and there is also a specified target state for the strategy automaton, that automaton will be also set forward on that target state. Otherwise, the strategy automaton remains unchanged. Overall the algorithm can be described by the following pseudo code sequence:

```
$q_L$ := start state of learning unit automaton
$q_S$ := start state of strategy template automaton
notVisitedStrategyItems := set of all strategy items

// choose candidate with best assessment
loop {
  // compute set of candidate transitions, assessments and associated strategy transitions
  (candidates, assessment[candidates], strategyCandidate[candidates])
     := assessment($q_L$, $q_S$, notVisitedStrategyItems);
  if (candidates = {}) { // allow remaining navigation according to unit automaton
    finishNavigation();  //        until a finite state is first reached
    exit;
  }

  // prepare navigation options
  nextItem[candidates]
    := getFirstTransitionsOnPathToCandidates(candidates);
  associateOptimalTargetWithNextNavigation(assessment[candidates],
                                           nextItem[candidates]);
  optionallyPresentAdditionalNavigationTargets();

  // wait for the user to decide for one option
  chosenCandidate := waitForUserSelection(nextItem[candidates]);

  // execute navigation to next knowledge item
  navigateTo(nextItem[chosenCandidate]);

  // execute next step for learning automaton and eventually for strategy automaton
  $q_L$ := nextItem[chosenCandidate];
  // if we really reach the targeted item forward the strategy automaton accordingly
  if (q_L == candidate)
     q_S := strategyCandidate[chosenCandidate];

  updateDataSet(notVisitedStrategyItems); // update according to navigation
}
```

Due to space limitations we can only sketch the main idea of this algorithm. However, the details of it are just about proper technical implementation and do not alter the idea itself.

Running Example. When applying the strategy template to the learning unit (using knowledge type based correspondence ≈) and following always the recommended navigation, the navigation algorithm will sequence the course as k_O, then k_E, k_R in arbitrary order, then k_{A1}, and finally k_{A2}.

6 A Graphical representation

Both, a learning unit and a strategy template can be represented as a graph, where graph nodes correspond with knowledge items and edges correspond with relations.

For the remainder of this section, we make the following assumptions:

- Within learning units, there are 2 types of relations, “before” and “belongs to” the first of which is a directed one, the second an undirected one. Consequently they are represented as directed/undirected edges within the graph.

- Similarly within strategy templates, there are 2 types of relations, “didactically before” and “associated with” the first of which is a directed one, the second an undirected one. Consequently they are represented as directed/undirected edges within the graph.

- Undirected relations/edges have a transitive semantics, i.e. when 2 nodes A and B are related and B is also related with C, then A and C are also related.

- Any pair of nodes directly or indirectly related by an undirected edge must not be related by a directed edge.

These assumptions are analogously extended for strategy templates, where the introduced navigation algorithm is designed such that the relations specified in a learning unit take precedence over the relations defined in a strategy template.

6.1 Deriving a conditional regular expression from the graphical notation

The algorithm how to derive a conditional regular expression from the graphical notation (or more precisely from its associated graph structure) is very much the same for learning units and strategy templates. We start sketching it for strategy templates and then explain the differences for learning units.

Suppose a graph G=(N, E) describing a strategy templates, where N represents the nodes and E the edges within that graph. The directed edges from E are denoted as E^D, the undirected edges as E_U.

As first step, we construct a condensed node set N*, in which all nodes directly or indirectly connected via an undirected edge are melted together.

$N^* := \{[n] \mid n \in N \wedge [n] := \{m \in N : \exists\, k \geq 0\ \exists n_0,\dots,n_k \in N: \forall\, 0 \leq i \leq k: (n_i, n_{i+1}) \in E_U \wedge n = n_0 \wedge m = n_k\}\}$

Next, we construct regular expressions for each condensed node within N*. For each node $[n] \in N^*$ we assume $[n] = \{n_0,\dots,n_k\}$. Then, we use the permutation to build *regexp*([n]) := $<n_0,\dots,n_k>$.

The additional condition is build upon all directed edges from G reaching into a node contained in [n]. It actually combines the visited attributes of the source nodes of the respective edges.

$$\textit{cond}\,([n]) := \bigwedge_{\forall (m,p) \in E^D : p \in [n]} v(m)$$

Finally, the overall regular expression for G* with nodes $[n_0],\dots,[n_k]$ is defined as

$$< \textit{cond}\,([n_0]): \textit{regexp}([n_0]), \dots, \textit{cond}\,([n_k]): \textit{regexp}([n_k]) >$$

The same algorithm can be applied in order to transform learning units with one exception. The conditional expression for condensed nodes as well as the overall expression is not built upon the permutation (<…>) but on the combination ({…}*) of its elements.

The expression power of regular expressions is much higher than for the presented graphical representation. Other graphical presentations or transformation could be introduced to allow for describing more complex learning units or strategy templates.

Running example. An example for both a graphical representation and the derived regular expression has already been given in Sections 2 and 3.

7 Summary

This paper presents a new approach for supporting pedagogical guidance in e-learning courses by the means of didactic strategies. The approach differs from existing ones as it supports a completely graphical definition of learning strategies which can be automatically applied to arbitrary course content. This is underpinned by a uniform formal representation of learning strategies and course structures.

The approach contains various degrees of freedom: Meta data annotated to course elements can be arbitrarily extended and then also considered in the various algorithms. Assessment functions can be modified to put more emphasis to specific pedagogical goals (e.g. optimizing the number of missed strategy items). Also the graphical representations can be adapted or enhanced to different environments. Finally, it's worth mentioning that the approach supports also well known strategies (e.g. [GA04]).

As a next step we intend to conduct a larger study in order to evaluate the acceptance of our approach with a larger range of course authors and pedagogical experts.

References

[ADL04] Advanced Distributed Learning Initiative: Sharable Content Object Reference Model, Version 2004, 2004. URL: http://www.adlnet.org/

[Ar04] Aroyo, L. et al.: EASE: Evolutional Authoring Support Environment. In (Lester, J. C. et al. Eds.): Proceedings of the ITS'04 Conference, Berlin: Springer Verlag, 2004, pp. 140-149

[Br01] Brusilovsky, P.: Adaptive hypermedia. User Modeling and User Adapted Interaction. In (Kobsa, A., Ed.): Ten Year Anniversary Issue 11 (1/2), 2001, pp. 87-110

[Br04] Brusilovsky, P.: KnowledgeTree: A distributed architecture for adaptive e-learning. In: Proceedings of The Thirteenth International World Wide Web Conference, WWW 2004 (Alternate track papers and posters), New York, NY, 17-22 May, 2004, ACM Press, 2004, pp. 104-113.

[Da05] Dagger, D. et al.: Personalisation for All: Making Adaptive Course Composition Easy. In IFETS journal of Educational Technology and Society, Special Issue on Authoring of Adaptable and Adaptive Educational Adaptive Hypermedia, 2005

[GA04] Gerteis, W.; Altenhofen, M.: From Didactics to Technology: Dynamic Course Profiles. In (Ehlers, U. et al., Eds.): E-Learning Services in the Crossfire: Pedagogy, Economy and Technology. BIBB Federal Institute for Vocational Training Publication, Bonn, 2004

[IM04] Inaba, A., & Mizoguchi, R.: Learning design palette: An ontology-aware authoring system for learning design. In: Proc. of the Int. conference on computers in education, Melbourne, Australia, 2004

[Ma89] Manber, U.: Introduction to Algorithms: A Creative Approach. Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1989

[Me99] Meder, N.: Didaktische Ontologien. In Proc. 6. Tagung der Deutschen Sektion der Internationalen Gesellschaft für Wissensorganisation, Hamburg, Germany, September 1999, pp. 401-416 (in German)

[Mo05] Monthienvichienchai, R. et al.: Discrepancies Between Reality and Expectation: Can Adaptive Hypermedia Meet the Expectations of Teachers?. In CELDA2005, Cognition and Exploratory Learning in Digital Age, Porto, Portugal, December 2005.

[Na05] Nabeth T. et al.: InCA: a Cognitive Multi-Agents Architecture for Designing Intelligent & Adaptive Learning Systems. In: special issue of ComSIS journal, 2:2, December 2005

[Sp02] Specht, M. et al.: Adaptive learning environment for teaching and learning in WINDS. In (Bra, Paul de, Hrsg): Adaptive hypermedia and adaptive web-based systems. Lecture notes in computer science, Springer, 2002, pp. 572 – 575

[Th02] Theilmann, W. et al.: Instructional strategies for collaborative e-learning. In Proc. of World Congress Networked Learning in a Global Environment, Berlin, Germany, May 2002, ICSC-NAISO Academic Press, 2002