

# A meta level to LAG for Adaptation Language re-use

Maurice Hendrix and Alexandra Cristea

The University of Warwick, Department of Computer Science  
Gibbet Hill Road, CV4 7AL, Coventry  
United Kingdom  
{maurice, acristea}@dcs.warwick.ac.uk

**Abstract.** Recently, a growing body of research targets authoring of content and adaptation strategies for adaptive systems. The driving force behind it is *semantics-based reuse*: the same adaptation strategy can be used for various domains, and vice versa. E.g., a Java course can be taught via a strategy differentiating between beginner and advanced users, or between visual versus verbal users. Whilst using an *Adaptation Language* (LAG) to express reusable adaptation strategies, we noticed, however, that: a) the created strategies have common patterns that, themselves, could be reused; b) templates based on these patterns could reduce the designers' work; c) there is a strong preference towards XML-based processing and interfacing. This has lead us to define a new meta-language for the LAG Adaptation Language, facilitating the extraction of common design patterns. This paper provides more insight into the LAG language, as well as describes this meta-language, and shows how introducing it can overcome some redundancy issues.

**Keywords:** LAG; AHA; Grammar; Educational Adaptive Hypermedia; Adaptation; Adaptation Engine.

## 1. Introduction

The use of adaptive systems [7] is increasingly popular. Commercial systems on the web (e.g., Amazon) or beyond (PDA device software) present at least a rudimentary type of adaptation. However, adaptation specification can not be fully expressed by standards<sup>1</sup> yet, and most commercial and non-commercial systems rely on proprietary, custom designed, system specific, non-portable, and non-interoperable adaptation. An intermediary solution, until standards emerge, is the creation of *Adaptation Languages*, which, with their power of semantics-based reuse, appear as a reliable future vehicle for all [8], [15]. Once written, the same adaptation strategy can be used for various domains. E.g., the strategy for *beginner-intermediate-advanced* written in the LAG language [8], could be used to teach students of varying knowledge level studying databases, mathematics or poetry. Similarly, the same domain model can be used with various adaptation strategies. E.g., a Java course can be taught via a strategy differentiating between beginner and advanced users, or between visual

---

<sup>1</sup> SCORM Simple sequencing allows basic adaptation. IMS-LD promises more for the future.

versus verbal users. However, there are a number of limitations regarding adaptation engines, which ultimately influence the efficient authoring of adaptation strategies, as based on an analysis of Interbook<sup>2</sup> [12] WHURLE<sup>3</sup> [14], AHA! [4], [5] and Personal Reader [1].

Thus, in this paper we define and analyze these limitations, illustrating them via a case study of a simple, yet powerful Adaptation Language, the *LAG language* [8]. Moreover, we propose a *meta-language*, as a supplement to LAG, showing how introducing it can overcome such limitations. Importantly, this solution is compatible with *extant* adaptation engines, instead of requiring the creation of new engines.

## 2. Adaptation Engine Issues and Limitations

The following are issues and limitations identified as influencing the authoring flexibility of adaptive hypermedia (AH) systems:

- L1. Most adaptive hypermedia delivery systems determine the *adaptation on a per-concept base* [1]. A broad knowledge of the whole content at every adaptation step is (usually) unavailable, mainly due to run-time complexity limitations. Thus, adaptation strategies cannot specify complex inter-concept rules; e.g., a strategy with an arbitrary set of labels denoting topics of interest, displaying to the user concepts related to his topic, without limiting the possible topics at design-time.
- L2. Adaptation engines don't (usually) allow for *non-instantiated program variables* [1]. Thus, authoring strategies which involve an unknown number of types, categories, etc., are currently not permitted. All domain-related variables need to be instantiated in the authoring stage.
- L3. There are extreme difficulties arising when *combining multiple strategies* [1]. Adaptation engines usually update sets of variables based on some triggering rules, without knowing which high-level adaptation strategies these variables represent. An example of a combined strategy currently difficult to implement is one where the system checks whether the user prefers text or images, and then displays the preferred type of content, filtered via a beginner-intermediate-advanced strategy, where concepts are shown based on the user's knowledge.

In AHA! [4], [5] reasoning is mainly done on a per-concept base (for persistent attributes). Volatile attributes can contain expressions, which reference other attributes, allowing for backward reasoning. However, this does not fix problem L1 entirely. This method only allows for access to variables concerning concepts that have already been visited before or are in the same line of hierarchy. AHA! also does not allow for any free program variables (L2). AHA! can combine strategies (via the LAG language [8]) but does not offer any solution to conflicting naming (L3).

InterBook<sup>4</sup> [12] uses a knowledge-based approach to create adaptive, interactive electronic textbooks. Adaptation is more limited than in AHA!: it uses a classification of domain concepts into a *spectrum* and allows for adaptation towards the user's

---

<sup>2</sup> <http://www.contrib.andrew.cmu.edu/~plb/InterBook.html>

<sup>3</sup> <http://whurle.sourceforge.net/>

<sup>4</sup> <http://www.contrib.andrew.cmu.edu/~plb/InterBook.html>

current knowledge state. The prerequisites are computed on a per-concept base, and neither free variables nor combined strategies are at all possible (L1-L3).

In WHURLE<sup>5</sup> [14], the *lesson plan* specifies a path through the content *chunks*. Rules are defined on a per-concept base (L1), and no free program variables are allowed (L2) [13]. Multiple strategies are possible by using XML pipelines [16]. The issue of different strategies using conflicting naming (L3), however, remains.

Personal Reader [1] can deal with more sophisticated issues. It uses an RDF ontology with complex reasoning, so limitation L1 does not apply. However, it still does not offer free program variables (L2). Combining rules in an RDF ontology is less problematic, as multiple relationships can be defined at the same time. There are however limitations as to what can be implemented efficiently. For example, if we look at the OWL<sup>6</sup> ontology language (based on RDF), we see that although OWL Full is complete and has no limitations as to what can be expressed, only the very limited set of OWL Lite can be implemented efficiently. This however comes at the cost of a greater computational complexity, and therefore leads to a less scalable system.

### 3. A Case Study

#### 3.1 The Theoretical Framework, in short: the LAOS Framework

In order to analyze the LAG language [8], a short briefing about the underlying theory is necessary. The LAG Adaptation Language instantiates the adaptation layer of the LAOS model [10]. The LAOS model is a general layered framework for Adaptive Hypermedia authoring, containing five layers: *Domain Model (DM)*: with domains of content and their relations; *Goal & Constraints Model (GM)*: filtering useful domain concepts (possibly from *multiple* domains) and grouping them; *User Model (UM)*: with user specific variables, e.g. level, age, etc.; *Adaptation Model (AM)*: defining how the content is adapted to users' needs; *Presentation Model (PM)*: determining look & feel, navigation elements, as well as quality of service parameters.

#### 3.2 Expressing Static Content in CAF (Common Adaptation Format)

The LAG Adaptation Language [8] processes information stored in CAF (Common Adaptation Format) [11]. CAF is an interfacing format that describes the static data needed for describing a Goal & Constraints Model (GM), and all the Domain Models (DM) it uses, ensuring that they all conform to LAOS [10]. Thus it defines concept maps, concepts, links and resources that are to be used in adaptation. CAF is mainly targeted at improving interoperability between different Adaptive Hypermedia systems, by offering a way to represent data in a system-independent way; e.g., CAF can be used to transport a GM and its related DMs between MOT [9] and AHA! [4]. CAF represents these models using a relatively simple XML format (see below).

---

<sup>5</sup> <http://whurle.sourceforge.net/>

<sup>6</sup> <http://www.w3.org/TR/owl-ref/>

```

<?xml version="1.0"?>
<!DOCTYPE CAF SYSTEM 'CAF.dtd'>
<CAF>
  <domainmodel>
    <concept>
      <name>Relational databases</name>
      <concept>
        <name>Fundamentals</name>
        <attribute>
          <name>theory</name>
          <contents>Relational theory</contents>
        </attribute>
      </concept>...
    </concept>...
  </domainmodel>
  <goalmodel>
    <lesson>
      <contents weight="0" label="beginner_title">
        Relational databases\Fundamentals\title</contents>
      <contents weight="0" label="beginner_text">
        Relational databases\Fundamentals\theory</contents>
    </lesson>
    <lesson>
      <contents weight="0" label="intermediate_title">
        Relational databases\Fundamentals\Definition\title
      </contents>...
    </lesson>...
  </lesson>
</CAF>

```

The example shown above represents a CAF file that contains one GM and one DM which this GM uses. The DM is called *Relational databases*. This Domain Model has one domain concept called *Fundamentals*. This concept has a domain attribute *theory* with the contents *Relational theory*. It also has other attributes, omitted due to lack of space. We see that the GM uses both the *title* and *theory* attributes of the *Fundamentals* concept of the *Relational databases* DM. It sets weights and labels for them, which, as we will see in section 3.3, are used by LAG [8] adaptation strategies. In short, titles and other elementary information (not shown here) are displayed to beginner students, and the theory is displayed to more advanced students.

### 3.3 A strategy in the LAG Adaptation Language

The Adaptation Language (LAG) [8] can express reusable adaptation strategies, describing adaptation, as prescribed by the Adaptation Model of LAOS [10]. As seen in section 3.2, items in the Goal & Constraints Model (GM) have weights and labels, which are used by the adaptation strategies. Below we show an example Adaptation Strategy described in the LAG language. This language works on structures defined by CAF, and thus is *domain specific*, with its domain being adaptive hypermedia in general; at the same time, within adaptive hypermedia, it is *generic*, as it can work with any content domain (e.g., databases, neural networks; chemistry, etc.).

The example below illustrates a simple strategy called '*beginner – intermediate - advanced*'. This strategy displays concepts to the user, depending on his experience level. The example uses the simpler labels 'beg', 'int' and 'adv' for concepts intended for beginner, intermediate and advanced users respectively (instead of the labels 'beginner\_title', 'beginner\_text', etc., as in section 3.2). The example also uses a number of variables. The 'show' variable, which determines if the concept is to be shown, is one of the few *core set variables* of the LAG language. Other variables are used, e.g., to record if a concept has been visited, or how many concepts of a particular group of concepts have been visited. It is more elegant to keep the set of variables as small as possible. Fewer variables make strategies smaller in terms of file size, thus easier to read, and in terms of memory usage, thus performing better.

The initialisation part (below) is performed only the first time the user enters the system; after that, every time the user selects a (lesson) concept, the implementation part (see comment 6), describing the actual interaction loop, is performed.

**initialisation (**

1) general: make every general (unlabeled) concept readable; mark every concept as "not visited yet" (beenthere =0);

```
while (true) (
  PM.GM.Concept.show = true
  UM.GM.Concept.beenthere = 0 )
```

2) initialize the number of concepts for beginning to advanced students to 0;

```
UM.GM.begnum=0 UM.GM.intnum=0 UM.GM.advnum=0
```

3) count and store the actual number of concepts for beginner students;

```
while GM.Concept.label == beg ( UM.GM.begnum += 1 )
while (GM.Concept.label == beg) ( PM.GM.begnum +=1 )
```

4) count and store the actual number of concepts for intermediate students;

```
while (GM.Concept.label=int) (
  PM.GM.Concept.show = false
  UM.GM.intnum +=1 )
```

5) count and store the actual number of concepts for advanced students;

```
while (GM.Concept.label == adv) (
  PM.GM.Concept.show = false
  UM.GM.advnum += 1 )
```

6) set the level of the student to beginner, for the first entry in the system;

```
UM.GM.knowlvl = beg )
```

**implementation (**

7) UM.GM.Concept.beenthere computes the "number of times a Concept has been accessed". The following keeps track of how many beginner, intermediate and advanced concepts still need to be visited. These rules are checked each time a concept is accessed. One concept is not 'aware' of other concepts, however.

```
if (UM.GM.Concept.Access==true) then (
  if (UM.GM.Concept.beenthere = 0) then
    if (GM.Concept.label == beg) then(
```

```

        UM.GM.begnum--1      )
    if (GM.Concept.label == adv) then(
        UM.GM.advnum--1      )
    if (GM.Concept.label ==int) then(
        UM.GM.intnum--1      )
    UM.GM.Concept.beenthere+=1 )

```

8) Change the stereotype from beginner to intermediate; from intermediate to advanced when appropriate; make relevant concepts visible;

```

    if enough(UM.GM.begnum < 1
        UM.GM.knowlvl==beg,2) then(
        UM.GM.knowlvl = int      ) )
    if enough(UM.GM.begnum < 1
        UM.GM.knowlvl==int,2) then(
        UM.GM.knowlvl = adv      ) )
    if (GM.Concept.label == UM.GM.knowlvl) then(
        PM.GM.Concept.show = true ) )

```

The strategy above illustrates a classical case of adaptation, to students of varying knowledge level<sup>7</sup>. The strategy works well because it ‘knows’ what labels to expect in the CAF file representing the Goal & Constraints model: ‘beg’, ‘int’, ‘adv’. Currently, if other labels are also present, the conversion ignores them. However, what happens if we want to represent strategies with more complex labels, such as the ones in section 3.2 There, we had, e.g., various labels starting with ‘beginner\_’ or ‘intermediate\_’, but we didn’t know in advance how many types of such labels exist. Still, we should expect to be able to perform some adaptive strategy and express it in the form of an adaptation program. As variables need to be instantiated, this introduces an intermediate step in the processing, as the next section shows.

#### 4. Solutions to Adaptation Engines Issues and Limitations

Previously (section 3.3), we have seen an illustration of two of the current limitations listed in section 2: (L1) concept-based adaptation, where the same rule has to be copied in all concepts, and one concept doesn’t (normally) affect other concepts directly, and (L2) the fact that adaptation engines don’t allow for non-instantiated variables. A straightforward way of defeating these problems would be to build new adaptation engines. The first scenario could be achieved by establishing which labels exist, in the initialisation step. The second issue could be overcome by either allowing arrays of labels, or otherwise allowing multiple data to be stored in the label. However, in order to function with *current* systems, these issues should be solved in the authoring stage. For the third limitation (L3), the difficulty in application of multiple strategies, the MOT to AHA! converter, e.g., has already implemented an elegant solution (unique to our knowledge so far), in that it can apply multiple LAG files, with different adaptation strategies, with the order of execution set by priorities of the respective strategies (1: highest priority; any following number: lower priority):

---

<sup>7</sup> For examples of strategies please visit: <http://prolearn.dcs.warwick.ac.uk/strategies.html>

```
priority x /* where x is a number */
```

Nevertheless, this method could override previous variables (e.g., if two strategies use UM.Concept.knowledge, only the update of the highest priority strategy counts). Thus, a *unitary strategy merge*, keeping track of all variables in use, based on multiple labels for domain-related concepts and attributes, is preferable. Moreover, many types of variables (e.g., arrays) are not allowed by Adaptation Languages, due to lack of adaptation engine support, limiting the adaptation that can be expressed.

However, we have noticed that a) strategies have common patterns, as has already been shown previously in [3], that could be reused; b) templates based on these patterns could reduce the designers' work; c) there is a strong preference towards XML-based processing and interfacing. Thus, XML-based templates should be used to move the processing to the authoring side and facilitate the extraction of re-use patterns.

For the creation of LAG files based upon a LAG template, explicit knowledge about the content is needed. CAF represents a flexible format for Adaptive Hypermedia content and is also used by AHA!. Therefore, a CAF file will be our choice for the content. For the LAG template files, the LAG files which follow the extended LAG description introduced in section 3.3 will be required. A pre-processor can replace the constructs added in section 5 by traditional LAG constructs. The resulting LAG file will then describe the same adaptation behaviour as the template LAG file, but for the specific labels encountered in the CAF lesson.

Implementing the pre-processor as a web-based application enables it to transfer both the unchanged CAF file as well as the resulting LAG file to the AHA! system, provided, of course, the appropriate rights are set and the pre-processor is on the same system as AHA! (currently AHA! only allows uploading files through a Java tool). To facilitate the use of multiple strategies, it should be possible to select multiple LAG templates. The user should be given a choice between creating the AHA! lesson and downloading the resulting LAG file. This process could, if the direct lesson creation is used, smoothly replace the current process, without requiring any extra effort from the user. This process is shown in the figure below.

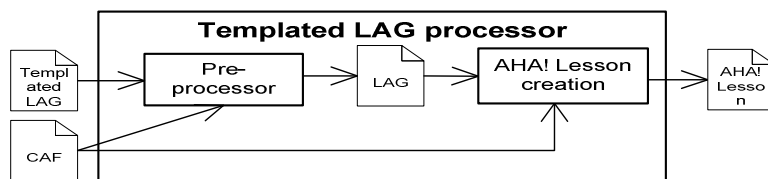


Fig. 1. System setup of template LAG Pre-processor.

## 5. Meta-level addition to LAG

To solve the limitations mentioned in section 2, we add, as said, a *pre-processing step* to the whole authoring process. This step takes a LAG template and the content, in the

form of a CAF file, and pre-processes it. The result is a new LAG file which extends the strategy sketched by the LAG template for the specific content described in the CAF file. We want to accommodate future changes to LAG, as well as have our approach be reusable and easily implemented and maintained. Therefore we propose an XML-based notation for the template LAG files, while keeping the original LAG language unchanged for compatibility with current systems. Note that alternatively the changes could be incorporated into the Lag language directly but then it would lose its compatibility with existing systems. Since CAF is already written in an XML based notation, both documents can be used as input for an XSLT transformation which generates the resulting LAG file. Below we give the DTD (document type definition) for the template LAG file.

```
<!ELEMENT TLAG ((LAGfragment*, LIKE*)*)>
<!ELEMENT LIKE attribute CDATA value CDATA
(LAGfragment, MATCH, LAGfragment, (LAGfragment*, LABEL,
LAGfragment*)*) >
<!ELEMENT LAGfragment (#PCDATA)>
<!ELEMENT MATCH EMPTY>
<!ELEMENT LABEL EMPTY>
```

A template LAG file consists of a number of blocks of the following kind: a number of *LAG fragments* followed by a *LIKE* block. The LAG fragments contain LAG adaptation program snippets, similar to the examples showed in section 3.3. The *LIKE* blocks consist of an attribute and a regular expression against which it is matched, followed by a fragment of LAG program. The word *MATCH* represents the place where the *LABEL* needs to match the regular expression.

Below we show a fragment of the beginner-intermediate-advanced strategy. It shows how template LAG can be used to create an adaptation strategy that works with the CAF file example in section 3.2:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Server SYSTEM "tlag.dtd">
<TLAG> ...
  <LAGfragment>UM.GM.beginner_number= 0 </LAGfragment>
  <LIKE attribute='GM.Concept.label' value='*beginner*'>
    <LAGfragment> while(UM.GM.label= </LAGfragment>
      <MATCH/>
      <LAGfragment>) (UM.GM.beginner_number+=1)</LAGfragment>
    </LIKE> ...
</TLAG>
```

Following is an extract of the result of the pre-processing of a LAG template and the CAF file of the earlier example. The complete result is a LAG file, tailored towards the content of the CAF file. In the snippet below we see that the variable *UM.GM.beginner\_number* is increased by one for each variable using the label *UM.GM.label.beginner\_title* or *UM.GM.label.beginner\_text*. These were exactly the labels matching the regular expression *\*beginner\**. Applying (the DTD of) the LAG template solves some of the problems mentioned in section 2.

```
...
while
```

```
(UM.GM.label.beginner_title||UM.GM.label.beginner_text.  
) (UM.GM.beginner_number+=1) ...
```

- L1. *Problem: adaptation on a per-concept base*; a broad knowledge of the whole content at every step of the adaptation is (usually) unavailable.  
*Solution:* such knowledge is not necessary in the adaptation engine. It is acceptable that this type of knowledge can be acquired as a one-off, at authoring time, as it is not to be expected that content labels will change at execution time. Therefore, the authoring strategy should contain this knowledge. As for an author it is difficult to manually extract all the pedagogical label types existent in a course, templates such as the DTD of the template LAG above can help in dealing with groups of labels (such as all labels containing ‘beginner’, i.e., ‘\*beginner\*’). An author can then generate the appropriate adaptation strategy (of which a snippet is shown above) in an easy and quick manner, making use of existing patterns in the authoring strategy itself.
- L2. *Problem:* adaptation engines don’t usually allow for *non-instantiated program variables*.  
*Solution:* Unknown domain-related variables can be instantiated in the authoring stage, with the help of patterns specified via the LAG template language based on the above DTD. It is not necessary for an author to perform these searches manually; the two-step authoring system can extract unknown variables for him.
- L3. *Problem:* the extreme difficulties arising when *combining multiple strategies*.  
*Solution:* similar pattern extraction mechanisms have to be used in order to merge adaptation strategies. In (nearly) every system there is a limited number of weights and labels; this causes problems in combining a number of strategies greater than the number of weights and labels available. A solution to this can be to apply pattern matching on labels in order to be able to encode multiple strategies, by using the same label field. This thus enhances simple prioritization of strategies, as it allows the combination of multiple strategies which each requires specific labels.

## 6. Conclusions and further work

In this paper we have analyzed adaptation problems inherent in current adaptation engines, which reduce the power and generality of Adaptation Languages. We described and exemplified these issues with the help of the *LAG language*, currently one of the only exchange formats of Adaptation Language specification between systems. Moreover, we have moved one step further, by proposing improvements that can overcome run-time issues of adaptation engines, by solving them at the authoring stage. More specifically, templates can be used to create adaptation strategies, customized for the given domain models and pedagogical labels. For this purpose, we have proposed the *template LAG language*. The process is technically implemented by adding a pre-processor to the system setup, which has access to content at compile-time, which is not available at run-time. In such a way, more powerful adaptation strategies can be created for *existing* adaptation engines.

**Acknowledgments.** This research has been performed with the help of the EU ALS Minerva project (Adaptive Learning Spaces, 229714-CP-1-2006-NL-MINERVA-M) and the PROLEARN Network of Excellence.

## References

1. AHA! Adaptive Hypermedia For All, <http://aha.win.tue.nl>
2. Abel, F., Brunkhorst, I., Henze, N., Krause, D., Mushtaq, K., Nasirifard, P., Tomaschewski, K.: Personal Reader Agent: Personalized Access to Configurable Web Services. ABIS 2006 - 14th Workshop on Adaptivity and User Modeling in Interactive Systems, Hildesheim, October 9-11 (2006)
3. Brown, E., Cristea, A., Stewart, C., and Brailsford, T. Patterns in Authoring of Adaptive Educational Hypermedia: A Taxonomy of Learning Styles, International Peer-Reviewed On-line Journal "Education Technology and Society", Special Issue on Authoring of Adaptive Educational Hypermedia, (Volume 8, Issue 3). (2005)
4. De Bra, P., Aerts, A., Berden, B., De Lange, B., Rousseau, B., Stanic, T., Smits, D., Stash, N.: AHA! The Adaptive Hypermedia Architecture. Proceedings of the ACM Hypertext Conference, Nottingham, Uk, August, 81--84 (2003)
5. De Bra, P., Smits, D., Stash, N., The Design of AHA!, Proceedings of the ACM Hypertext Conference, Odense, Denmark, August 23-25, 2006 pp. 133, and <http://aha.win.tue.nl/ahadesign/>, 2006.
6. Brailsford, T.J., Stewart, C.D., Zakaria, M.R. & Moore, A. (2002). Autonavagation, Links and Narrative in an Adaptive Web-Based Integrated Learning Environment. 11th International World Wide Web Conference, Honolulu, Hawaii, 7-11 May 2002.
7. Brusilovsky, P.: Adaptive hypermedia, User Modelling and User Adapted Interaction, Ten Year Anniversary Issue, (Alfred Kobsa, ed.) 11 (1/2), 87--110 (2001)
8. Cristea, A.I., Calvi, L.: The three Layers of Adaptation Granularity. UM'03, Pittsburg, US. (2003)
9. Cristea, A.I., De Mooij, A.: Adaptive Course Authoring: My Online Teacher. Proceedings of ICT'03, Papeete, French Polynesia (2003)
10. Cristea, A.I., De Mooij, A.: LAOS: Layered WWW AHS Authoring Model and their corresponding Algebraic Operators. WWW03 (The Twelfth International World Wide Web Conference), Alternate Track on Education, Budapest, Hungary (2003)
11. Cristea, A.I., Smits, D., De Bra, P.: Writing MOT, Reading AHA! - converting between an authoring and a delivery system for adaptive educational hypermedia -, A3EH Workshop, AIED'05, Amsterdam, The Netherlands (2005)
12. Eklund, J., Brusilovsky, P.: InterBook: An Adaptive Tutoring System UniServe Science News Vol. 12. March 1999. 8--13 (1999)
13. Kostelník, R., Bieliková, M., Web-Based Environment using Adapted Sequences of Programming Exercises. In Proc. of Information Systems Implementation and Modelling - ISIM 2003. M. Beneš (Ed.). MARQ Ostrava, Brno, April 28-30, pp.33-40.
14. Moore, A., Stewart, C.D., Zakaria, M.R., Brailsford, T.J.: WHURLE - an adaptive remote learning framework, International conference on Engineering Education (ICEE-2003), July pp. 22-26, Valencia, Spain (2003)
15. Stash, N., Cristea, A.I., De Bra, P., Adaptation Languages as vehicles of explicit intelligence in Adaptive Hypermedia, In International Journal on Continuing Engineering Education and Life-Long Learning, vol. 17, nr 4/5, pp. 319-336, InderScience, 2007.
16. Zakaria, M.R., Moore, A., Stewart, C.D., Brailsford, T.J. (2003). "Pluggable" user models for adaptive hypermedia in education. Proceedings of the Fourteenth ACM Conference on Hypertext and Hypermedia, August 26-30, 2003, Nottingham, UK. pp 170-171