LAOS: Layered WWW AHS Authoring Model and their corresponding Algebraic Operators

Alexandra I. Cristea
Faculty of Computer Science and Mathematics
Eindhoven University of Technology
Postbus 513, 5600 MB Eindhoven, The Netherlands
+31-40-247 4350
a.i.cristea@tue.nl

Arnout de Mooij
Faculty of Computer Science and Mathematics
Eindhoven University of Technology
Postbus 513, 5600 MB Eindhoven, The Netherlands
a.m.d.mooij@stud.tue.nl

ABSTRACT
In this paper, we describe the design steps for WWW authoring of adaptive hypermedia via a five layer model. We argue that we need to introduce the *goal and constraints* model between the *domain model* and *adaptation* and *user models*, in order to be able to generate adaptive hypermedia on the fly and to actually implement the so often quoted re-usage paradigm. We also show the operators necessary to implement functionality at the different levels, and exemplify this layered construction with MOT, an adaptive hypermedia (in particular, courseware) authoring system we have built at the Eindhoven University of Technology.

Categories and Subject Descriptors
H.1 [**Information Systems**] Models and Principles; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia - *architectures, navigation, user issues*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing - *abstracting methods, dictionaries, indexing methods*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - *clustering, information filtering, query formulation, relevance feedback, retrieval models, search process, selection process*; E.1 [**Data**]: Data Structures - *distributed data structures, graphs and networks*; K.3.1 [**Computers and Education**]: Computer Uses in Education - *distance learning*

General Terms
Design, Experimentation, Standardization, Languages, Theory.

Keywords
Adaptive authoring, adaptive hypermedia, AHS, AHAM, ontologies, semantic web, RDF, MOT

1. INTRODUCTION
Adaptive hypermedia is a relatively new field, tracing back to the early 1990s. Adaptive hypermedia system (AHS) are becoming nowadays more popular, due to their correlation with the recent strive of the W3C and the IEEE LTTF [18] community towards (ontology-based) customization and the semantic Web [28]. The success of such research AHS as AHA! [15], Interbook [7], TANGOW [9] or other Web adaptation engines such as Firefly (before it was bought by Microsoft) has pushed AHS forward. Their edge over classical Intelligent Tutoring Systems (ITS) systems [6] relies on their simplicity: they contain a simple domain model, user model (usually an overlay model of the domain model), aimed at a quick response, which is extremely beneficial in the speed-concerned WWW environment. However, for quite a long while there has been a lack of powerful authoring tools for adaptive hypermedia [5][11]. One of the main reasons was the great (but fruitful) diversity in AHS implementations, many with implicit models [31]. Recently, stimulated by the ripening of the field, a group of researchers is working towards the implementation of adaptation standards [12][15], which can stay at the basis of such authoring systems. This lead to a strive towards obtaining clear explicit models for adaptive authoring [3][5][8][11][12][27][30][31].

Here we build upon AHAM [31], a well-known model developed at the Eindhoven University of Technology, and on previous models proposed by us for the educational field [11], to construct a more general layered model for adaptive hypermedia authoring.

The paper is organized as follows: Section 2 introduces our five layer model for AHS authoring. Section 3 populates the proposed model with algebraic operators and draws parallels to an RDF algebra. Section 4 exemplifies the defined model and operator implementations based on MOT, an AHS adaptive authoring system built at the Eindhoven University of Technology for on-line adaptive course production. Finally, section 5 draws conclusions by summarizing our contributions.

2. LAYERED MODEL
Previously we have defined a layered model for adaptive hypermedia authoring design methodology for (WWW) courseware [11]. This model suggested the usage of the following main three layers: *conceptual layer* expressing the domain model (CL - with sub-layers: atomic concepts and composite concepts – with their respective attributes), *lesson layer* (LL - of multiple possible lessons for each concept map or combination of concept maps) and *student adaptation and presentation layer* (SAPL - based on: adaptation model and presentation model). All these layers should have been powered by the adaptation engine (AE). Note that already, compared to [27] we were using the lesson model (LM) as an intermediate model between the domain model (DM) and the user and adaptation model (UM, respectively AM).

Here we give a more generalized model for generic adaptive hypermedia authoring. The idea is based on the book–course or book–presentation metaphor: generally speaking, when making a presentation, be it on the Web or not, we base this presentation on

WWW 2003, May 20-24, 2003, Budapest, Hungary.
ACM

one or more references. Simplifying, a presentation is based on one or more books. With this in mind it is obvious why we cannot jump from the DM to the AM (or UM): it would be equivalent to skip the presentation and just tell the user to read the book. In other words, the search space is too big and there is a too high degree of generality (no purposeful orientation of the initial material - i.e., book).

Therefore, what we need is an intermediate authoring step that is *goal and constraints* related: goals[1] to give a focused presentation, and constraints to limit the space of the search[2]. Simplifying, we can consider the *goal* as being a specific end-state, and the constraint to be defined as a sub-layers of the GM model (see Fig. 1, where the GM is a multiple sub-layers model). So, in a general-purpose adaptive hypermedia authoring environment, LL is replaced by the *goal and constraints layer* (GM). Moreover, obviously, student adaptation and presentation returns to the *user model*, UM, and the teacher author becomes a *general adaptive hypermedia designer*.

There are some fundamental differences between having only DM or the two new layers, DM and GM, as follows:

- Dynamic (adaptive) presentation generation becomes possible [13].
- The actual presentation seen by the user can contain both elements of the GM as well as elements of the DM (e.g., for clarification of an explanation based on only the GM, the other elements/ objects of the respective concept, or the other concepts related to the current concept, can be referred, via a jump over one layer).
- This increases the flexibility and expressivity of the created adaptive presentations.
- The AE has to actually implement not only *selectors*, but also *constructors* [27], as presentations can contain any type of combination of (ordered and weighted) attributes of concepts; in AHAM constructors are mentioned, but considered outside the scope of the model.
- This however increases the complexity of the system, and issues such as guaranteeing *termination* and *confluence* get new dimensions [27].

The total model is composed therefore of five components: DM, GM, UM, AM, PM, as can be seen in Fig. 1.

Moreover, we defined in previous research [11] some (concept map oriented) design steps for the authors to take, with regard to the first layered authoring model introduced. Below is a new refinement of these steps, reflecting the requirements imposed by the new layered model:

- STEP 1: write concepts + concept hierarchy
- STEP 2: define concept attributes (define main and extra attributes)
- STEP 3: fill concept attributes (write contents)
- STEP 4: add content related adaptive features regarding GM (design alternatives – AND, OR, weights, etc.)
- STEP 5: add UM related features (simplest way, tables as in AHAM [30], with attribute-value pairs for the user-related entities)
- STEP 6: decide among adaptation strategies, write in adaptation language medium-level adaptation rules (such as defined in [8]) or give the complete set of low level rules [12] (such as condition-action (CA [31]) or IF-THEN rules).
- STEP 7: define format (presentation means-related; define chapters)
- STEP 8: add adaptive features regarding presentation means (define variable page lengths, variables for figure display, formats, synchronizations points [29], etc.).

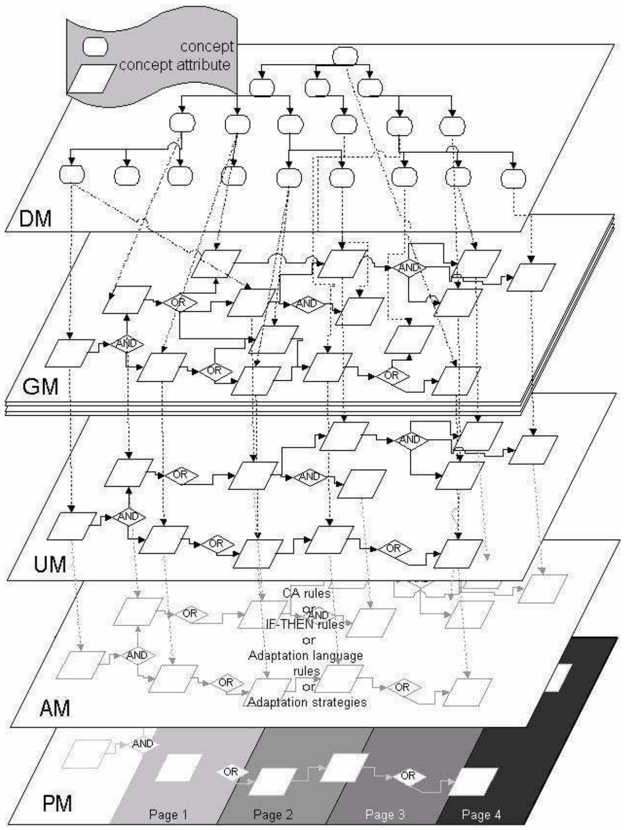

Figure 1. The five level AHS authoring model.

In the following we will analyze what type of operators we need for the authoring process of each layer.

3. ALGEBRAIC OPERATORS PER LAYER

3.1 Conceptual Layer

At the conceptual layer level we have a set of basic operators that follow basically the ones defined in [3]. The main difference here

[1] By introducing goals it is also clear why this level is a dense level made of multiple versions for each initial concept map or combination of concept maps: simply because there are multiple design goals to consider.

[2] Note that this still means that various flexibility degrees are left for the adaptation to the user and presentation model, so that the presentation material doesn't become uniquely determined.

is that we do not deal with tasks, but with goals and constraints. Goals are more general than tasks and include them and their practical aspects, but can be (and are) also more abstract. Moreover, the algebraic operators here have to reflect the new refined model structure.

First we have to give a more formal definition of the concept map elements (*objects*)[3].

Definition 1. We consider a concept map *CM* of the AHS to be determined by the tuple <C,L>, where C represents the set of concepts and L the set of links ($CM \subseteq \mathbf{CM}$, the set of all concept maps of the AHS).

Definition 2. A concept $c \in C$ is defined by the tuple $<A_c, C_c>$ where A_c ($A_c \neq \emptyset$) is a set of attributes and C_c a set of sub-concepts.

Definition 3. A_{min} is the minimal set of (standard) attributes required for each concept to have ($A_c \supseteq A_{min}$).

This minimal set of standard attributes is determined by the adaptive course design constraints, that aim at creating concepts annotated with sufficient meta-data, as prescribed by W3C for the creation of the semantic web [28]. Note that if $A_{min} = \emptyset$ this means that there are no required standard attributes.

Definition 4. A concept $c \in C$ is a composite concept if $C_c \neq \emptyset$.

Definition 5. A concept $c \in C$ is an atomic concept if $C_c = \emptyset$.

Definition 6. A link $l \in L$ is a tuple $<c1, c2, n_l, w_l>$ with $c1 \in C$, $c2 \in \mathbf{CM}.C$ *start* and *end* concepts, respectively, n_l a name or label of the link and w_l a weight of the link.

This means that links can be added between any concept of the owned *CM* as the *start* concept to any concept of the whole **CM** space of concepts. If the end concept is outside the current *CM*, the author will not be allowed to edit the contents of the end concept. Please note that at this level these weights' meaning is only given by the semantics of their label.

Definition 7. An attribute $a \in A_c$ is a tuple *<var,val>*, where *var* is the name of the attribute (variable or type) and *val* is the value (contents) of the attribute[4].

Constraints on the model:

Definition 8. Each concept *c* must be involved at least in one link *l*. This special relation is called *hierarchical link* (or link to father concept). Exception: root concept.

As all the sets above are finite, they can be given (relative) identification numbers. Therefore, concept *c* is determined (and therefore can be referred to) by its identification $i \in \{1,\ldots,C\}$ (where C=card(C)) and the attributes of concept *i* are $a_i[h]$, with $h \in \{1,\ldots,A_i\}$ and $A_i \geq A_{min}$ (where $A_i = card(A_c)$ and $A_{min} = card(A_{min})$).

With the above domain definitions, we need to define algebraic *operators* and the respective *operations* over the model. The justification of the need of constructing a proper algebra for the AHS authoring model is given on one hand by the motivation towards comparable semantics of AHS authoring systems [17], and on the other hand by the need of allowing a crisp structuring of the authoring process. The algebraic operators are of four types: *constructors* (create, edit), *destructors* (delete), *visualization* or extractors: (list, view, check) and *compositors* (repeat). From the perspective of their effects, they can be categorized as being: *restructuring* (constructors, destructors and any compositors using at least one operator belonging to the previous categories) or *structure neutral* (visualization and any compositors applied to visualization alone). The complete *operation – operator* list is presented in Table 1.

Table 1. Algebraic operators definitions for DM authoring

operation & operator	**Range of operation in DM**	**Description**
Create & **'C'**	• **Input (*atomic*)**: optionally object *name* (text label) of objects such as for CM_x,; father concept for *c*,; ids (numerical) of (*c1*, *c2*) and *expression* for *l*, a_i[h] (with h>A_{min}) • **Input (*set*)**: as above for sets of objects $\{c_j\}^+, \{l_j\}^+, \{a_i[h]\}^+$ (with $1 \leq h \leq A_{min}$) • **Output space**: **CM**, C, L, A_c • **Output**: CM_x , $\{c_j\}^*, \{l_j\}^*, \{a_i[h].var\}^*$	• creates one object such as a concept map, concept, link, a non-standard attribute • creates sets of objects such as set of new hierarchical child nodes and/ or links connected to the same parent or a full standard attributes set
Edit & **'E'**	• **Input**: object ids or *expression* • **Output**: { *{CM_x, c, l, a_i*[h]*}.val*}*	edits the object value[5]
Delete & **'D'**	• **Input**: as the two above together, *condition* or *expression* • **Output space**: **CM**, C, L, A_c	deletes an object (set) from the corresponding structure or empties the contents
List & **'L'**	• **Input**: Any sets from above, optional *condition* or *expression* • **Output**: *interface object*	lists the objects of the set(s)
View & **'V'**	• **Input**: (set of) object id-s and mode (e.g., *Graph/ Text*) • **Output**: *interface object*	gives alternative views of the results to the author
Check & **'Ck'**	• **Input**: (set of) object id-s from **CM**, C, L, A_c, *checking goal*, (and implicitly their *value domains*) • **Output**: *interface object*	checks the *checking goal* for the selected object and informs about value domain trespasses
Repeat & **'R'**	• **Input**: Any of above, number of times or other stopping *condition* • **Output space**: same as operation performed	Repeats any of the operations above

The *condition* is a statement with a truth-value attached or a Boolean function that works on objects in the **CM** space and constants, uses atomic operators, comparison operators (<, ≤, =, ≥, >, or the equivalent string operators) between literals and logical operators (*and*, *or*, *not*).

[3] All these elements defined below are considered to be indexed.

[4] With values being volatile or not according to AHAM [30].

[5] We assume here that *val* is defined analogously for *CM*, *c*, *l*.

The *expression* represents (set of) objects of the **CM** space or the result of applying an operator. An expression allows the composition of the operators according to their domain restrictions.

The *interface objects* are texts, figures, multimedia presentations, any combinations of objects, etc., for the authoring environment. Note that they might be different from the interface objects for the adaptive hypermedia end-user.

These operators we have defined very often work, in fact, on databases, due to the fact that the DM and GM, in their CM form, can be easily represented as databases, as we will be illustrating in section 4. Therefore it might be useful to replace the operators with their database counterpart. As the Resources Description Framework (RDF) [4][20] is intended to serve as a metadata language for the WWW, we have compared our algebraic operators with a RDF database-based algebra (Table 2).

Symbols used: π projection; σ selection; $\times$ join; $\bowtie$ natural join; $\cup$ union; $\cap$ intersection; — difference.
Due to lack of space we have not written the details of the full expressions of the RDF database-based algebra counterpart.

Table 2. RDF algebra database counterpart of atomic operators[6]

DM operator	RDF database-based algebra counterpart [17][7]	Comparison: limitations, advantages
'C'	Node [name[8], id_superconcept]() Link[[name],c1, c2](object: expression)	No attribute creation in RDF algebra (can be implemented as node creation, but CM semantics is lost)
'E'	No current counterpart	
'D'	No current counterpart	
'L'	π[name](object)=L(object.name) (object set1)$\times$(object set2) = L(os1$\times$os2) os1$\cup$os2=L(os1,os2) os1$\cap$os2=L(os1,os2,os1.c$\neq$os2.c) os1—os2=L(os1, os1.c$\neq$os2.c) os1$\bowtie$ [condition]os2= =L(os1$\times$os2, cond)	*List* is a more general operator, that can extract any information provided with a condition
'V'	σ["Text"](object set)=V("Text", object set)	*Selection* is more general than *View*, which is presently limited to 2 types.
'Ck'	σ[Goal](object set)=Ck(Goal, object set)	as above
'R'	Map[f](expression)=R(f, expression) Kleene Star: *[f](expression)=R(f, expression)	*Repeat* cannot normally implement infinite loops, like *Kleene Star* (could be done via a *condition* with constant truth value)

[6] Note that there is only a limited equivalence, depending on the input structure, and our operators are in principle more general.

[7] Slightly modified for comparison

[8] Id-s we consider to be automatically generated and unique. Names can be repeated, to keep ontological mappings easy.

This comparison however shows clearly that, although it is undoubtedly useful to make the link to the internal database structure of this type of representation, and also the link to the RDF architecture, our model needs more expressivity and flexibility than is offered by these basic models.

3.2 Goal and Constraints Layer

Some of the operators at the GM level (Table 3) can be (almost) transferred directly from the DM level (Table 1), but we have to take into consideration the insertion of AND/OR relations and the extra constraints introduced. Moreover, OR relations combine their elements according to weights[9]. However, there is also a drastic change in structure: there are (practically) no predefined sets of standard attributes to include in a goal-oriented presentation, and every concept has to point to an attribute from the CM.

These types of restrictions form the constraints of the layer, thus generating a smaller search space. The combination of AND-OR relations is supposed to lead to the goal of the layer.

First we have to give a more formal definition of the goal map elements (*objects*)[10]. We consider a goal map *GM* of the AHS to be a special *CM*, as follows.

Definition 9. A concept $c \in \mathsf{C}$ in *GM* is defined by the tuple < A_c, C_c> where A_c (card(A_{min})=2) [11] is a set of attributes and C_c a set of sub-concepts.

Definition 10. A link $l \in \mathsf{L}$ in *GM* is a tuple <$c1$, $c2$, n_l, w_l> with $c1 \in \mathsf{C}$, $c2 \in$ **CM**.C[12] *start* and *end* concepts, respectively, n_l a name representing the type (i.e., hierarchical or AND/OR connections) of the link and w_l a weight of the link.

Table 3. Atomic algebraic operator definitions for GM authoring

Atomic operation & operators	Range of operation in GM	Description
Create & **'C'**	• **Input**: original concept id in *CM* and attribute id; optionally object *name* (text label) of objects such as for GM_x, father concept for *c*; ids (numerical) of (*c1*, *c2*); *expression* for *l* • **Input**: as above for sets of objects $\{c_j\}^+$,$\{l_j\}^+$,$\{a_i[h].var\}^+$ ($1 \leq h \leq 2$) • **Output space**: **CM**, C, L, A_c • **Output**: GM_x, $\{c_j\}^*$,$\{l_j\}^*$, $\{a_i[h].var\}^*$	• creates one object such as a goal and constraints map, concept, link, a non-standard attribute • creates sets of objects e.g., set of new hierarchical child nodes +/- links to the same parent or a full standard attributes set

[9] The exact way of combining the weights has to be set by the triple (UM,AM,AE).

[10] All these elements defined below are considered to be indexed.

[11] Each *GM* concept has only 2 attributes: '*name*' and '*contents*'.

[12] Links can be added between any concept of the owned *GM* to any concept of the whole **CM** space of concepts, within *GM* or jumping a level, to the DM.

Edit & **‘E’**	• **Input**: object ids or *expression* • **Output**: { *{GM$_x$, c, l, a$_i$*[h]*}.val*}*	edits the object value[13]
Delete & **‘D’**	• **Input**: as the two above together, *condition* or *expression* • **Output space**: **CM**, C, L, A_c	deletes an object (set) from the corresponding structure or empties the contents
List & **‘L’**	• **Input**: Any sets from above, optional *condition* or *expression* • **Output**: *interface object*	lists the objects of the set(s)
View & **‘V’**	• **Input**: (set of) object id-s and mode (e.g., *Graph/ Text*) • **Output**: *interface object*	gives alternative views of the results to the author
Check & **‘Ck’**	• **Input**: (set of) object id-s from **CM**, C, L, A_c, *checking goal*, (and implicitly their *value domains*) • **Output**: *interface object*	checks the *checking goal* for the selected object and informs about value domain trespasses
Repeat & **‘R’**	• **Input**: Any of above, number of times or other stopping *condition* • **Output space**: same as operation performed	Repeats any of the operations above

The *CM* constraints are respected by the *GM*.

Note that only at this level AHAM [30] can be applied, and that this happens in the special case where the links’ *end* concepts are in C (*c1,c2*∈C). This is because AHAM does not allow to combine *attributes* (in AHAM notation, *fragments*) that are belonging to (originating in) different concepts, thus implying a very rigid adaptation space.

3.3 User, Adaptation and Presentation Model

UM and AM have been described relatively well by AHAM [30].

However, a maybe more interesting way of representing the UM is to keep the conformity with the DM and GM (uniform ontological representation [20]) and to also represent the UM as a concept map (CM). In such a way, relations between the variables within the UM can be explicitly expressed as relations in the UM, and do not have to be “hidden” among adaptation rules. A table of attribute-value pairs cannot show any relation that might exist between the different UM variables. Of course, if the UM happens to be just an overlay model of the DM, this type of linked representation results implicitly (via concept links).

We have introduced in [12] a new three-layer adaptation model (defining *low level assembly-like adaptation language*, *medium level programming adaptation language* and *adaptation strategies language*) that we are in the process of refining and populating, but this is beyond the scope of the present paper.

The PM has to take into consideration the physical proprieties and the environment of the presentation and provide the bridge to the actual code generation for the different platforms (e.g., HTML, SMIL [29]). Due to lack of space and to the fact that PM is so platform oriented, we are not going to go into details about this model here. For our purpose it is only important to note that the consideration about PM should be kept separate from the ones for the other layers.

[13] We assume here that *val* is defined analogously for *GM, c, l.*

4. AN IMPLEMENTED EXAMPLE: MOT

In the following, we show for exemplification the definitions of the *Conceptual Layer* and *Goal and Constraints Layer* for a specific system developed at the Eindhoven University of Technology: the *MOT system*, an adaptive authoring system for adaptive hypermedia, previously described [13]. MOT is going to be used as extra reference material at the Faculty of Mathematics and Computer Science, Eindhoven University of Technology, for a 4th year undergraduate course on “Neural Networks”.

4.1 RDF Schema and Instance of MOT

4.1.1 RDF Schema of MOT

To continue with the RDF-mapping started in Table 2, we give next an RDF schema of an actual implementation of the DM and GM in MOT in Figure 2.

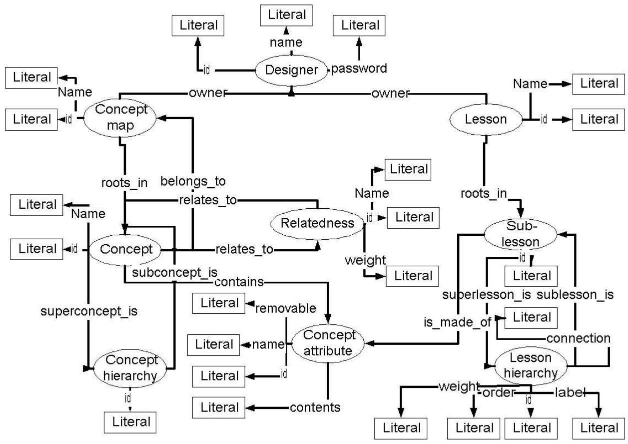

Figure 2. RDF Schema of MOT.

4.1.2 Domain Model

The structure of the DM can be seen in Figure 2, left hand side. In MOT, a concept contains one or more sub-concepts, which are concepts in their turn, hence inducing a hierarchical (tree) structure of concepts.

Each concept contains concept attributes. These attributes hold pieces of information about the concept they belong to. There are several kinds of attributes possible, corresponding to the different attribute instances in the diagram. For example, a concept can have a ‘*title*’-attribute, a ‘*description*’-attribute or an ‘*example*’-attribute.

Concept attributes can be related to each other. Such a relation, characterized by a label and a weight, indicates that their contents treat similar topics.

The hierarchical structure of concepts is implemented by means of a separate ‘concept-hierarchy’ entity, relating a super-concept to one / more sub-concepts. For re-usage and flexibility purposes, we allow sub-concepts to be only links to other concepts (so pointers to content instead of actual content). As a result, cycles can occur in the hierarchy. To prevent this, a check has to be performed, each time a hierarchy relation is added. I.e., a concept C_A in concept map A can link to a concept C_B in concept map B. If (a sub-concept of) concept C_B links back to concept C_A, a cycle appears. This kind of cycles (over one or more concept maps) are allowed, because course designers (teachers) should be able to link to each others concept maps unrestrictedly. However, this freedom can generate problems that will require a loop-checking mechanism in a future design and implementation step. For the present implementation, we assume that the course creation is done in such a way that unintentional loops are avoided.

Concepts can contain concept attributes. A concept attribute has been given a type (for example 'title' or 'text'). The relatedness of the concept attributes is replaced by a relatedness at concept-level. The relatedness of concepts is still based on commonalities between concept attributes. That is why a relatedness-relation is also given a type, indicating by which attributes the concepts are related. This type is one of the possible attribute types (for example '*title*', if the concepts are related by their titles).

A concept map couples a name and an owner to a hierarchy of concepts. It contains a pointer to the root of this concept hierarchy. The structure of this hierarchy is stored in several concept-hierarchy objects.

4.1.3 Goal and Constraints Model

The structure of the GM can be seen in Figure 2, right hand side. In MOT, the goals and constraints are given by lesson constructions. A lesson contains sub-lessons, which are lessons in their turn, hence creating a hierarchical structure of lessons. Sub-lessons within a lesson can be OR-connected (being lesson alternatives) or AND-connected. To facilitate this, a lesson contains a lesson attribute, which in its turn contains a holder for OR-connected sub-lessons or a holder for AND-connected sub-lessons. The holder contains the actual sub-lessons in a specified order.

A lesson attribute contains, besides the sub-lesson holders, one or more concept attributes. This is the link with the concept domain. The idea is that the lesson puts pieces of information that are stored in the concept attributes together in a suitable way for presentation to a student.

A lesson of a course is the equivalent of a concept map in the concept domain. It couples a name and an owner to a hierarchy of sub-lessons. It contains a pointer to the root of the sub-lesson hierarchy.

The hierarchy of sub-lessons consists of sub-lessons which are related by means of lesson-hierarchy objects, comparable to the concept-hierarchy objects in the concept domain. A sub-lesson which has no sub-lessons (e.g. is a leaf in the sub-lesson hierarchy) corresponds to a (one) concept attribute. This represents the link with the concept domain.

4.1.4 RDF Instance of MOT

Furthermore, Figure 3 shows and example RDF instance of MOT.

For the DM side (left hand side of Figure 3), we can see in the figure how concept *r11* is the root of the concept map *r2* owned by the designer *r1*. The concept *r4*, belonging to the same concept map is called "Discrete Neuron Perceptrons" and is a direct child of *r11*. Attribute *r9* called "Keywords" is contained in concept *r4* and contains the keyword list "perceptron; one-layer; multi-layer; weight; linear separability; perceptron convergence; boolean functions; region classifications in multidimensional space". Moreover, concept *r4* is related to concept *r12* via the attribute "Keywords" in a proportion of 24%.

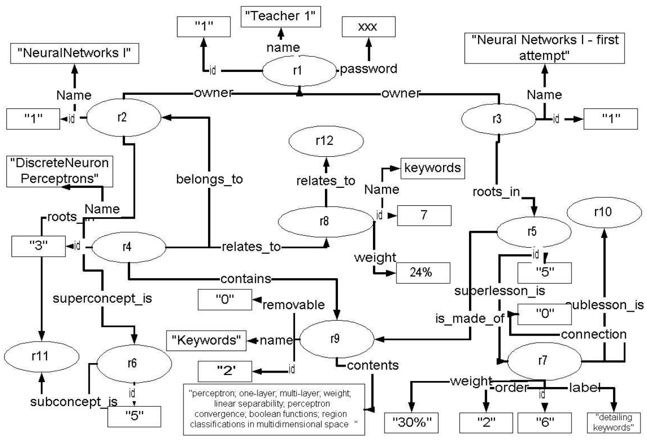

Figure 3. RDF Instance of MOT.

For the GM side (right hand side of Figure 3), the figure shows us that the previously mentioned attribute *r9* expressing the "Keywords" of concept *r4* is assembled in sub-lesson *r5*, which is also the root of the GM lesson model. Lesson *r5* also contains sub-lesson *r10* in an OR connector (connection="0") with the weight 30%, the priority order "2" and the label "detailing keywords".

In this way, specific instances of MOT can be represented in RDF.

4.2 CM and GM as Databases in MOT

To show how the CM and GM can be implemented with the definitions above, we show the composing elements of the MOT system. These are the statements to create the database tables of MOT (Figures 4,5). The database implementation follows in principle the RDF Scheme in Figure 2.

So, MOT justifies basing AHS authoring algebra on databases.

4.3 Run-time WWW Operations in MOT

The interface is based on the interface of the existing My Online Teacher system [23]. This means for one thing that it is a web interface based on CGI-scripts written in the Perl language. In principal the interface consists of two parts, reflecting the two parts of the RDF-schema diagram (Figure 2): one part for *designing concept maps* and one for *designing lessons*.

In MOT a teacher logs in via a login-screen with password check. S/he then enters a menu where s/he can choose between the concept maps and/or lessons s/he has already created. S/he can also select to create a new concept map or lesson.

- After selecting a concept map (Figure 6), the concept map frameset will appear. This frameset consists of two frames. On the left hand side the concept map structure is displayed and on the right hand side information about the selected concept (attributes) is shown.
- After selecting a lesson (Figure 7) from the menu, the lesson frameset will appear. This frameset also consists of two frames. On the left hand side the lesson structure is displayed and on the right hand side information about the selected sub-lesson is shown.

The specific operations with the concept map corresponding to the DM and the lesson map corresponding to the GM can be followed in the two Figures 6,7. They implement at a higher level the 'C', 'E', 'D', 'L', 'V', 'Ck', and 'R' operators (tables 1,3).

CREATE TABLE Teacher
(

Id	INTEGER PRIMARY KEY	Unique number.
Name	TEXT NOT NULL	Teacher's name.
Password	TEXT NOT NULL	Teacher's password.

);
CREATE TABLE Concept
(

Id	INTEGER PRIMARY KEY	Unique number.
Owner	INTEGER NOT NULL	Owner (creator) of concept. References Teacher.
Timestamp	TEXT	Not used.
Mapid	INTEGER NOT NULL	Map to which concept belongs. References Concept map.

);
CREATE TABLE ConceptAttribute
(

Id	INTEGER PRIMARY KEY	Unique number.
Concept Id	INTEGER NOT NULL	Concept to which attribute belongs. References Concept.
Standard Attribute Id	INTEGER NOT NULL	Standard attribute type or 100 (if not). References StandardAttribute.
Name	TEXT NOT NULL	Attribute name, if it is not a standard attribute.
Contents	TEXT NOT NULL	Attribute contents.

);
CREATE TABLE Conceptmap
(

Id	INTEGER PRIMARY KEY	Unique number.
Name	TEXT NOT NULL	Conceptmap name.
Owner	INTEGER NOT NULL	Owner (creator) of conceptmap. References Teacher.
Rootconcept Id	INTEGER NOT NULL	Root concept of conceptmap, which is a tree of concepts. References Concept.

);
CREATE TABLE StandardAttribute
(

Id	INTEGER PRIMARY KEY	Unique number.
Name	TEXT NOT NULL	Standard attribute's name.

);
CREATE TABLE ConceptmapAttribute
(

Id	INTEGER PRIMARY KEY	Unique number.
Conceptmap Id	INTEGER NOT NULL	Conceptmap that has this attribute as a standard attribute. References Conceptmap.
Standard attributeId	INTEGER NOT NULL	Standard attribute that is included in this concept map. References StandardAttribute.
Include	INTEGER NOT NULL	1 = include in lesson (when converting to a lesson), 0 = do not include in lesson.

);

Figure 4. Concept Map in MOT.

CREATE TABLE ConceptHierarchy
(

Id	INTEGER PRIMARY KEY	Unique number.
ConceptId1	INTEGER NOT NULL	Parent concept in relation. References Concept.
ConceptId2	INTEGER NOT NULL	Child concept in relation. References Concept.

);
CREATE TABLE Relatedness
(

Id	INTEGER PRIMARY KEY	Unique number.
ConceptId1	INTEGER NOT NULL	References Concept.
ConceptId2	INTEGER NOT NULL	References Concept.
Name	TEXT NOT NULL	Name of relation.
Weight	DOUBLE NOT NULL	Weight of relation.
Type	INTEGER NOT NULL	Relation type, which corresponds to a standard attribute. References table StandardAttribute.

);
CREATE TABLE AllKeywords
(

Id	INTEGER PRIMARY KEY	Unique number.
ConceptId	INTEGER NOT NULL	Concept to which the keyword belongs. References Concept.
Keyword	TEXT NOT NULL	Keyword contents.

);
CREATE TABLE Lesson
(

Id	INTEGER PRIMARY KEY	Unique number.
Name	TEXT NOT NULL	Lesson's name.
Owner	INTEGER NOT NULL	Owner (creator) of lesson. References Teacher.
ToplessonId	INTEGER NOT NULL	Root sub-lesson of lesson tree. References Sublesson.

);
CREATE TABLE Sublesson
(

Id	INTEGER PRIMARY KEY	Unique number.
AttributeId	INTEGER NOT NULL	Concept attribute in which the contents of this sub-lesson is stored. References ConceptAttribute

);
CREATE TABLE LessonHierarchy
(

Id	INTEGER PRIMARY KEY	Unique number.
Sublesson Id1	INTEGER NOT NULL	Parent sub-lesson in relation. References Sublesson.
Sublesson Id2	INTEGER NOT NULL	Child sub-lesson in relation. References Sublesson.
Connection	TEXT NOT NULL	'AND', if child sub-lesson is part of a sequence (or stand-alone), or 'OR', if child sub-lesson is one out of more alternatives.
Orderind	INTEGER NOT NULL	Order index that indicates the position of the child sub-lesson relative to the other sub-lessons of the parent sub-lesson.
Weight	DOUBLE	Weight of hierarchy relation.
Label	TEXT	Label/name of hierarchy relation.

);

Figure 5. CM (cont.) and Lessons in MOT.

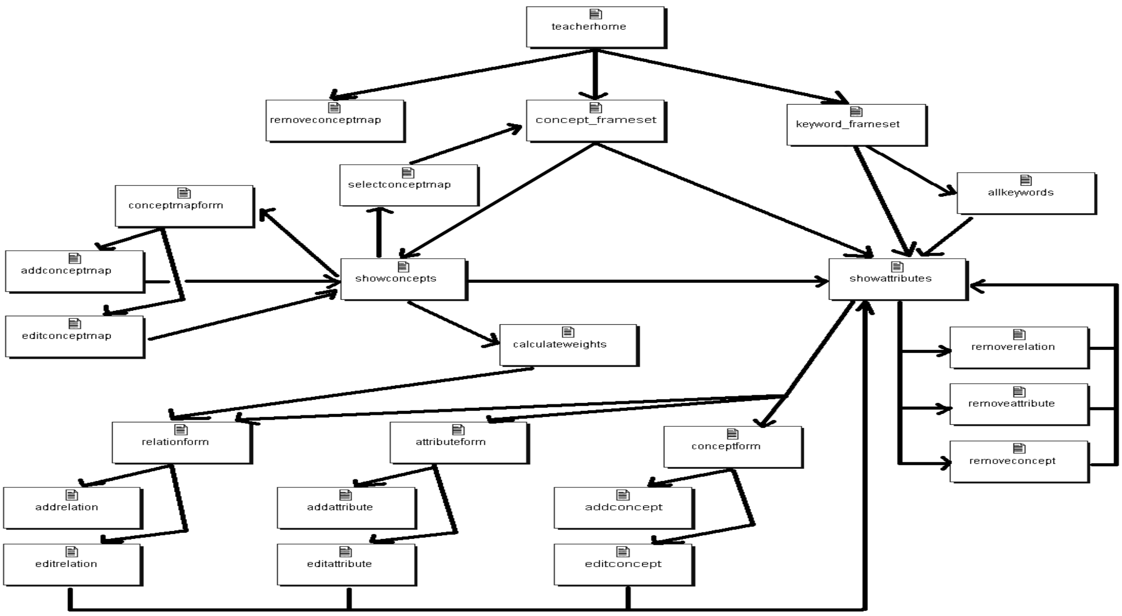

Figure 6. Call graph for the cgi-files of the concept map part.

The operations in Figure 6 are based on the definitions in Table 1 and the operations in Figure 7 on those in

Table 3. There are two connections between the concept map frameset and the lesson frame set, as follows.

- When the user is working in the concept map frameset, s/he can choose to edit/convert the existing concept map to a lesson, deciding on what attributes to keep and which to ignore. The result will be a lesson with a hierarchical structure following the pseudo-order of the *concept – sub-concept* relations and the pseudo-order of their respective attributes.
- When the user is working in the lesson frameset, s/he can choose to add a sub-lesson based on a concept attribute. S/he then will be presented with the concept map-frameset, where s/he can select a concept map, a concept and finally a concept attribute to add to the lesson. After this, s/he is redirected back to the lesson frameset.

The concept map structure, as well as the lesson structure, are displayed as trees resembling the tree structure for showing directory structures in, for example, the Microsoft Windows operating systems (i.e., as lists containing sub-lists).

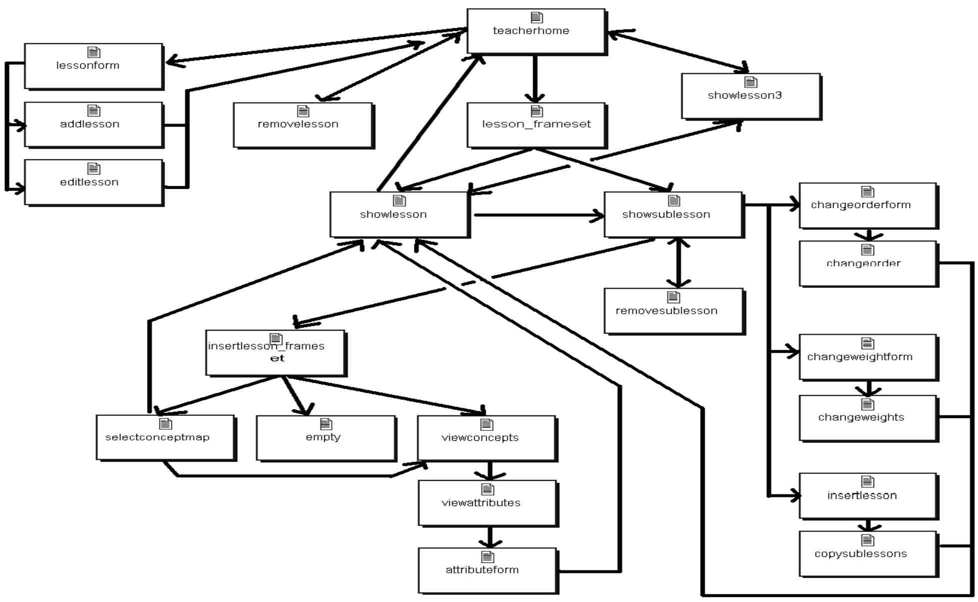

Figure 7. Call graph for the cgi-files of the lesson map part.

An element in a concept map or lesson can be moved or selected by pressing the appropriate hyperlink attached to it.

4.4 IMPLEMENTATION NOTES

4.4.1 Database

The database is implemented using MySQL, which is a freely distributed SQL database. Some advantages of MySQL are: it is for free; it is the most popular and widely distributed SQL database; it is easy to use.

However, MySQL is very limited in some aspects. Important features that are missing in MySQL are: Views, Functions and procedures and Table constraints.

MySQL supports only a very limited number of table constraints. For example, it is not possible to add a constraint to a table that demands a certain field to reference another table.

PostgreSQL is another freely distributed SQL database, which does have all of the above features. It should therefore be taken into consideration for future implementations to use this database instead of MySQL. The SQL statements that are used in the current MOT system should also work with PostgreSQL, in the worst case requiring some slight syntactical modifications.

4.4.2 Client-Server Structure

The MOT interface uses CGI scripts. The CGI (Common Gateway Interface) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. CGI scripts are processed by the web server, to transmit information to the database engine, receive the results and display them to the client. A CGI script can be interpreted by the web server directly, in contrast to a CGI program (for example written in C++) that would have to be compiled first.

To transfer parameters from one script to another two methods exist. With the GET method, parameters are passed after a question mark in the URL. With the POST method, parameters are passed hidden to the user. Both methods are used in MOT. When the user presses a hyperlink to go to another page, parameters are passed using the GET method. These parameters are visible in the location bar of the web browser. The values entered by the user in the several fill-in forms are passed using the POST method.

Luckily, a great Perl CGI library, CGI.pm [10], exists, that hides all kinds of technical aspects of the CGI to the programmer. In MOT, functions from this library are used most of the time when calling the CGI. An extra advantage of this is that it makes the code easier to read.

For the database communications, functions from the Perl DBI library are used. This library provides a database independent interface for Perl, which means that the code would still work if the database should be replaced by some other database. This library also makes the code easy to write and read.

Furthermore, for most of the rest of the processing, the Perl language is used. Perl [25] is a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal).

The fact that Perl is optimized for scanning arbitrary text files makes it very useful for the calculation of relatedness relations (which are automatically generated links [13]). For this task a lot of occurrence counts are needed, which can be very efficiently programmed in the Perl language. However, these very efficient constructs are not as easy to read.

4.4.3 Other User-side Interface Issues

The concept map and lesson structures are displayed as nested lists. At first, non-collapsible HTML-lists were implemented. However, these lists tended to grow very large, making it hard for the user to keep a good overview. Also it didn't make sense to send calls to the server each time the user wanted to increase or decrease the view granularity (operator 'V'). That is why collapsible lists were introduced, using JavaScript. The JavaScript collapsible lists are taken from [19].

5. CONCLUSIONS

In this paper we introduced a five level AHS authoring model with a clear cut separation of the processing levels:

1. the *domain model* (DM),
2. the *goal and constraint model* (GM),
3. the *user model* (UM),
4. the *adaptation model* (AM) and finally
5. the *presentation model* (PM).

Compared to previous models we have introduced a *goal and constraints* level and its corresponding model between the domain model and the user and adaptation models.

We have delimited the actions that take place at each level first informally, than with a higher degree of formalism, focusing especially on the newly refined layers, DM and GM.

We defined the objects of the model and described primitive algebraic operators to work on them. These operators are based on a RDF database oriented algebra [17] and on our previous research on defining operations for a slightly different domain [3]. In order for our set of algebraic operators to be sufficient (and to form an algebra) it would have to be complete, covering any possible transactions that occur in an AHS authoring setting.

Moreover, we have showed an implementation of the proposed model for MOT, an adaptive hypermedia system WWW authoring environment being developed at the Eindhoven University of Technology. The motivational aspect about ways in which MOT confers benefits to users (teachers) is treated in [13].

For the specific case of MOT, we have presented the RDF schema and an example instance for describing the system, as well as the database table definitions for the focus issues, the DM and GM.

The main justification of introducing the GM lies in the dynamic adaptive presentation possibilities is opens. MOT already implements some primitive functionality of automatic transformations from the DM to the GM (described elsewhere [13]) that lead us to claim to work towards "a course that writes itself" for the specific application of adaptive WWW courseware.

6. ACKNOWLEDGMENTS

This research is linked to the European Community Socrates Minerva project "Adaptivity and adaptability in ODL based on ICT" (project reference number 101144-CP-1-2002-NL-MINERVA-MPP).

7. REFERENCES

[1] 2L690: Hypermedia Structures and Systems, Lecturer: Prof. De Bra. http://wwwis.win.tue.nl/~debra/2L690/

[2] Apache 1.3 download and documentation. http://httpd.apache.org/docs/

[3] Aroyo, L., Cristea, A.I., and Dicheva, D. A Layered Approach towards Domain Authoring Support. In Proceedings of ICAI 2002 (Las Vegas, US) CSREA Press.

[4] Brickley D., and Guha, R.V. Rdf vocabulary description language 1.0: Rdf schema. W3C Working Draft 30 April 2002. http://www.w3.org/TR/rdf-schema/.

[5] Brusilovsky, P. Adaptive hypermedia, User Modeling and User Adapted Interaction, Ten Year Anniversary Issue (Alfred Kobsa, ed.) 11 (1/2), 2002, 87-110.

[6] Brusilovsky, P., Schwarz, E., Weber, G. ELM-ART: An intelligent tutoring system on world wide web. In Proceedings of International Conference on Intelligent Tutoring Systems (ITS'96) (Montreal, Canada, June 1996), 261-269.

[7] Brusilovsky, P., Eklund, J., and Schwarz, E. Web-based education for all: A tool for developing adaptive courseware. Computer Networks and ISDN Systems, In Proceedings of Seventh International World Wide Web Conference (14-18 April 1998) 30 (1-7), 291-300.

[8] Calvi, L., and Cristea, A.I. Towards Generic Adaptive Systems Analysis of a Case Study. In Proceedings of AH'02 (Malaga, Spain, May 2002) Adaptive Hypermedia and Adaptive Web-Based Systems, LNCS 2347, Springer, 79-89.

[9] Carro, R. M., Pulido, E. Rodríguez, P. Designing Adaptive Web-based Courses with TANGOW .In proceedings of the 7th International Conference on Computers in Education, ICCE'99 (Chiba, Japan, November 4 - 7, 1999) V. 2, 697-704.

[10] CGI.pm – a Perl5 CGI Library. http://stein.cshl.org/WWW/software/CGI/

[11] Cristea, A.I., and Aroyo, L. Adaptive Authoring of Adaptive Educational Hypermedia, In Proceedings of AH 2002, Adaptive Hypermedia and Adaptive Web-Based Systems, LNCS 2347, Springer, 122-132.

[12] Cristea, A.I., and De Bra, P. Towards Adaptable and Adaptive ODL Environments. In Proceedings of AACE E-Learn'02 (Montreal, Canada, October 2002), 232-239.

[13] Cristea, A., De Mooij, A. Adaptive Course Authoring: MOT, My Online Teacher. In Proceedings of ICT-2003, IEEE LTTF International Conference on Telecommunications, "Telecommunications + Education" Workshop (Feb 23 - March 1, 2003 Tahiti Island in Papetee - French Polynesia) (in press).

[14] Cristea, A.I., Okamoto, T., and Kayama, M. Considerations for Building a Common Platform for Cooperative & Collaborative Authoring Environments. In Proceedings of AACE E-Learn'02 (Montreal, Canada, October 2002), 224-231.

[15] De Bra, P. and Calvi, L. AHA! An open Adaptive Hypermedia Architecture. The New Review of Hypermedia and Multimedia, vol. 4, Taylor Graham Publishers, 1998,115-139.

[16] European Community Socrates-Minerva project (project reference number 101144-CP-1-2002-NL-MINERVA-MPP). http://wwwis.win.tue.nl/~alex/HTML/Minerva/index.html

[17] Frasincar, F., Houben, G.J. , Vdovjak, R. , and Barna P. RAL: An Algebra for Querying RDF. In Proceedings of the 3rd International Conference On Web Information Systems Engineering (WISE 2002) (Singapore, December 2002).

[18] IEEE LTTF, Learning Technology Task Force. http://lttf.ieee.org/

[19] JavaScript collapsible list. http://devedge.netscape.com/toolbox/examples/2001/xbCollapsibleLists/

[20] Lassila, O. and Swick, R. R. Resource description framework (rdf) model and syntax specification. W3C Recommendation 22 February 1999. http://www.w3.org/

[21] Mizoguchi, R., Bourdeau, J. Using Ontological Engineering to Overcome Common AI-ED Problems, International Journal of AI in Education, 11 (2), 107-121.

[22] My English Teacher. http://wwwis.win.tue.nl/~alex/MyEnglishTeacher/TeachersSite/index.html

[23] My Online Teacher. http://wwwis.win.tue.nl/~alex/MOT01/TeachersSite-html/index.html

[24] MySQL documentation. http://www.mysql.com

[25] Perl documentation. http://www.perldoc.com or http://www.perl.com

[26] PostgreSQL documentation. http://www.postgresql.org/

[27] Wu, H., De Bra, P. Sufficient Conditions for Well-Behaved Adaptive Hypermedia Systems. In Proceedings of the First Asia-Pacific Conference on Web Intelligence: Research and Development (Maebashi, October 2001). Lecture Notes in Artificial Intelligence, Vol. 2198, Springer, 148-152.

[28] WC3, Semantic Web. http://www.w3.org/2001/sw/

[29] W3C, SMIL, Synchronized Multimedia Language. http://www.w3.org/AudioVideo/

[30] Wu, H., De Kort, E., De Bra, P. Design Issues for General-Purpose Adaptive Hypermedia Systems. In Proceedings of the ACM Conference on Hypertext and Hypermedia (Aarhus, Denmark, August 2001) 141-150.

[31] Wu, H. A Reference Architecture for Adaptive Hypermedia Applications, doctoral thesis, Eindhoven University of Technology, The Netherlands, ISBN 90-386-0572-2.