

An Adaptive Hypermedia Presentation Modeling System for Custom Knowledge Representations

Pablo Castells and José Antonio Macías
E.T.S. de Informática, Universidad Autónoma de Madrid
Campus de Cantoblanco, 28049 Madrid, Spain
+34-91-3482284, +34-91-3482241
{pablo.castells, j.macias}@uam.es
<http://www.ii.uam.es/{~castells, ~macias}>

Abstract. We propose a generic presentation system for adaptive educational hypermedia that is highly independent from domain knowledge representation and application state management. Our approach is based on providing a) a module for knowledge representation by means of the definition of domain ontologies that best fit specific domains and/or authors, b) a module for building courses by constructing semantic networks of interrelated domain ontology instances, and c) a presentation module where presentation models (templates and rules) are associated to ontology object classes and relations. By using an explicit presentation model, separate from course contents, course designers are provided with extensive control over the generation of all aspects of presentation, at a moderate development cost. Because minimum assumptions are made about how domain knowledge is structured and updated, our presentation system can be potentially integrated with a wide range of adaptive hypermedia support tools.

Keywords. Adaptive Hypermedia, Knowledge Representation, Ontology Engineering, Presentation Design, Web-Based Learning, User Modeling, Interface Design Tools.

1 Introduction

With the rapid introduction of web-based technology in the educational field, learners are gaining increasing autonomy, and instructional applications are reaching an unprecedented diversity of users. In this context, a growing interest has been raised for the development of hypermedia systems that are able to adapt automatically to different types of users, platforms and situation, and that take into account the evolution of each user over time [2]. A common implicit or explicit priority concern to all research work in adaptive education is that of finding an appropriate representation for pedagogical knowledge [8]. Each courseware development tool establishes its own way to structure the domain, so that designers must describe the subject matter as the tool prescribes, and the tool takes care of the selection, presentation and dynamic sequencing of teaching materials, and the interaction with the user.

In this paper we propose a generic hypermedia presentation system, PEGASUS (Presentation modeling Environment for Generic Adaptive hypermedia SUpport Systems), that makes minimum assumptions about how instructional knowledge is represented [6, 7]. It is our purpose to provide courseware designers with a simple specification paradigm for non-trivial adaptive presentation constructs, that can be used with different course management systems. In order to allow for different approaches, PEGASUS supports the definition of made-to-measure domain ontologies for the description and conceptual structuring of subject matter (as in [8]). Once an ontology is defined, designers build courses by creating domain objects and relating them together using the conceptual vocabulary defined by the ontology. Course presentation is designed by defining an explicit presentation model where presentations are associated to ontology object classes and relations.

2 Related Work

The explicit representation and use of semantic knowledge about a domain to facilitate or guide the access to information has been a primary concern in hypermedia systems from the early times [11]. In the literature on adaptive hypermedia many ways to structure knowledge have been proposed. Most of them are based on a level of contents, discretized on the basis of some kind of elementary unit, and a level of semantic structure, that is used as a road map to guide navigation. There is a great variation however as to how contents are structured, how the conceptual network is organized, and how both levels are connected.

To name a few examples, Interbook [1] structures courses into hierarchical aggregate units: *chapters*, *sections*, *subsections*, and *terminal pages*, accompanied by a set of interconnected *concepts* with two types of relation: *prerequisite* and *outcome*. The relative simplicity of this two-relation model contrasts with the lexical richness of other tools like HyperTutor [9], where the conceptual map takes a wide variety of relations from the educational theory literature: *prerequisite*, *sequence*, *aggregation*, *similarity*, *opposite*, *example*, *specialization*, and *exception*. AHA [5] allows a more flexible composition of pages, based on

conditional HTML fragments. Concepts have three boolean attributes to indicate the student's state of knowledge with respect to each concept: *known*, *read*, *ready to be read*, and relations between concepts have a parameter that represents a state (a boolean) or a quantitative measure (a number).

DCG [13] and TANGOW [3] are distinguished for generating the course structure, or parts of it, at runtime. In DCG a first level of interconnected concepts is defined, below which, for each concept, a tree of learning *tasks* and *subtasks* is created with the sequence of documents and actions to follow by the student to learn the concept [12]. Both task decomposition and the association of fragments to atomic tasks are determined dynamically at runtime by means of *rules*. TANGOW uses a conditional hierarchy similar to DCG where, unlike DCG, contents can also be associated to composite tasks which, as a consequence, can also be visited by the student. In other systems, the generation of semantic relations is even more dynamic and takes place by means of automatic search mechanisms based on metadata that are associated to information units [14]. This approach is useful when the knowledge space is too large and/or volatile to define and maintain explicitly the desired relationships.

Out of the hypermedia field, Eon [8] takes a more general approach than the preceding systems, allowing the author to define his/her own knowledge categories (*topics*), and the relations among them that s/he considers appropriate. Each topic is assigned sets of content units through different relations defined by the designer, such as *introduction*, *explanation*, *evaluation*, *basic level*, *advanced level*, or *summary*. The effective selection of contents takes place at runtime by applying predefined pedagogical strategies for content selection and ordering (e.g. choose an element randomly, or present all of them in sequence). Eon provides a graphical tool where the designer builds parameterized user interfaces, to which s/he associates units of contents, in such a way that interface widgets (buttons, tables, graphs, dialog boxes, etc.) take values from the knowledge unit being presented.

3 Knowledge Representation

Using a specific knowledge representation approach, like the ones described in the preceding section, a domain model is built. Many adaptive systems associate information about the student to domain model objects, in order to maintain an up-to-date model of the student's knowledge and goals with respect to the described subject matter (overlay model). This information is used at run-time to adapt the selection and presentation of contents and links to the user. While the cited systems provide for different forms of explicit author control over the teaching strategies (corresponding to conceptual map update mechanisms), the generation of pages, except in Eon [8], is done according to fixed presentation patterns and styles programmed into the tool.

Our system supports the automatic generation of hypermedia documents of the type supported by other adaptive systems, with full control for the designer over the visual aspects (presentation) of the generated hyperdocuments, and without imposing a particular representation of knowledge. To do so, like Eon, PEGASUS allows the definition of taxonomies made to fit the domain and/or the author. The terminology thus defined is used on the one hand for the description of the subject matter by the author, and on the other for the construction of presentation models associated to the different knowledge categories.

3.1 Domain Ontology

The domain ontology in PEGASUS consists of a set of classes that best fit a specific application domain or that reflect the specific view of a particular author on the domain. In our approach ontologies can be defined with a high degree of freedom, with very generic classes like Concept, Lesson, Fact, or more specific, like Algorithm, Theorem, or Definition, as the designer sees fit. Ontologies include terms for subject-matter information (e.g. a theorem has a statement and a proof), pedagogical information (e.g. lessons have levels of difficulty), and run-time (user and system) state information (e.g. whether a concept is known by the student). All this knowledge is captured by defining attributes for classes, and relations between classes.

PEGASUS provides a root class, *KnowledgeUnit*, and two predefined subclasses, *Topic* and *Fragment*, for ontology designers to subclass. *Topic*'s are presented to the end-user in a separate page, while several *Fragment*'s can be inserted in the same page. A predefined subclass of *Fragment*, *AtomicFragment*, carries content media (HTML source). *KnowledgeUnit* has a few predefined attributes like *id* and *title*, to which the designer can add others like *read*, *known* or *visible*, and new relations like *prerequisite* and *subunit*. Among other formats, PEGASUS allows the representation of ontology classes and domain instances in XML. The following example illustrates the definition of a class *Algorithm* with three relations: procedure, examples, and proof of correction. For the sake of brevity we omit here other relations that would normally be included, such as previous definitions, problem to solve, or analysis of complexity.

```
<Class name="Algorithm" parent="Topic">
  <Attribute name="recursive" type="Boolean"/>
  <Relation name="procedure" type="AtomicFragment"
    multivalued="false" title="Procedure"/>
  <Relation name="examples" type="AtomicFragment"
    multivalued="true" title="Examples"/>
  <Relation name="correction" type="Theorem"
    multivalued="true" title="Proof of Correction">
```

```

    <Attribute name="relevant" type="Boolean" />
    <Attribute name="difficulty" type="Number" />
  </Relation>
</Class>

```

Relations can have their own attributes, like the difficulty of the proof of correction in the above example, that reflect properties of the relation itself. All relations have a predefined *title* attribute that is used in certain cases to generate titles or text for hypermedia links.

In addition to a domain ontology, simpler data structures are defined by the designer to describe user profiles, information about the course (plan, goals, requirements, duration, number of students, etc.), platform characteristics and other aspects considered relevant for the adaptivity of the application being built.

3.2 Domain Model

Once an ontology has been defined, courses are constructed by creating semantic networks of domain objects, using the classes and relations defined in the ontology. The following example illustrates the creation of an instance of *Algorithm* to represent Dijkstra's algorithm for the shortest paths problem (we assume that the attribute *title* and the relation *prerequisite* are predefined in the root class *KnowledgeUnit*).

```

<Algorithm id="Dijkstra" title="Dijkstra's Algorithm" recursive="false">
  <prerequisites> <Algorithm ref="relaxation"/> </prerequisites> (1)
  <procedure>
    <AtomicFragment> <tt>Dijkstra(G,s)<br>&nbsp;&nbsp;&nbsp;Init(G,s)<br>&nbsp;&nbsp;&nbsp;Q = V[G] (2)
    <br>&nbsp;&nbsp;&nbsp;while Q not empty do<br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;u = ExtractMin(Q)<br>
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;for v in Adj[u] Relax(G,u,v)</tt> </AtomicFragment>
  </procedure>
  <examples> <AtomicFragment URL="exmpl.html"/> </examples> (3)
  <correction relevant="true" difficulty="0.6"> <Theorem ref="theorem1"/> </correction>
</Algorithm>

```

Elements with the attribute *ref* indicate references to other course units (for instance, line 1 refers to an algorithm with *id*="relaxation"). Atomic fragments can directly consist of a string, like Dijkstra's algorithm procedure above (line 2), or a web address, like in line 3.

At runtime PEGASUS maintains a copy of all domain objects for each user, where class attributes (e.g. *read*) are used to measure the user's progress. These values can be used to influence presentation (see Section 4), but a complementary update module is required to keep them up to date (see architecture, Section 5).

The domain model in PEGASUS supports the definition of adaptive elements in the model itself by means of the introduction of conditions on any part of the structure. For example, to select different examples depending on the student's experience level, line 3 would be changed to:

```

<examples>
  <test condition="user.expertise < 0.5"> <AtomicFragment URL="example1.html"/> </test>
  <test condition="user.expertise >= 0.5"> <AtomicFragment URL="example2.html"/> </test>
</examples>

```

This way it is possible to build dynamic structures like TANGOW [3] or DCG [13] task hierarchies, which take their definitive shape at runtime depending on the user model. Besides these adaptive elements, PEGASUS admits, though does not include by itself, any other mechanism for dynamic construction and modification of course structure. Our system takes care of how this may affect presentation, but how course structure and state are updated is external to the presentation system.

4 Presentation Model

Existing adaptive hypermedia systems miss an explicit presentation model. As a consequence, presentation is partly intermingled with contents (as in [5]), and partly set up automatically by the system according to rigid design choices (e.g. link annotation) that the designer cannot configure (see [1, 3] for instance). In PEGASUS, separation of content and presentation is achieved by defining a *presentation template* for each class of the ontology. Templates define what parts (attributes and relations) of a knowledge item must be included in its presentation and in what order, their visual appearance and layout. Templates are complemented with *presentation rules*, which are responsible for generating adaptive presentation constructs involving relations between domain objects from very succinct high-level descriptions given in templates. Whereas in Eon [8] user interface components are associated with specific units of knowledge, in PEGASUS presentations are defined for *categories* of knowledge.

4.1 Presentation Templates

Templates are defined by using an extension of HTML based on JavaServer Pages™ (JSP) [10], that allows inserting control statements (between `<%` and `%>`) and Java expressions (between `<%=` and `%>`) in the HTML code. In these templates, the designer can use all the presentation constructs of the HTML language (lists, tables, frames, links, forms, etc.), and insert, using very simple Java expressions, the domain items to be presented. For instance, a very simple template for class *Algorithm* could be as follows:

```

<h2> <%= title %> </h2>
<h3> Previous concepts </h3>
<%= prerequisites %>
<h3> Procedure </h3>
<%= procedure %>
<h3> Examples </h3> (4)
<%= examples %> (5)
<h3> Proof of Correction </h3> (6)
<%= correction %> (7)

```

In these templates the presentation author only needs to refer to attributes and relations of the presented class (shown in bold in the example). The presentation system takes care internally of aspects like automatically handling lists (multivalued relations like the examples of an algorithm), or recursively applying templates to referenced objects according to their class (e.g. the proof-of-correction *Theorem*'s of an algorithm). The resulting page for Dijkstra's algorithm with this presentation template can be seen in Figure 1. HTML elements surrounding the algorithm presentation (frame structure with contextual index on the left and *Previous / Next* buttons at the bottom) come from the presentation template for the root class *KnowledgeUnit*.

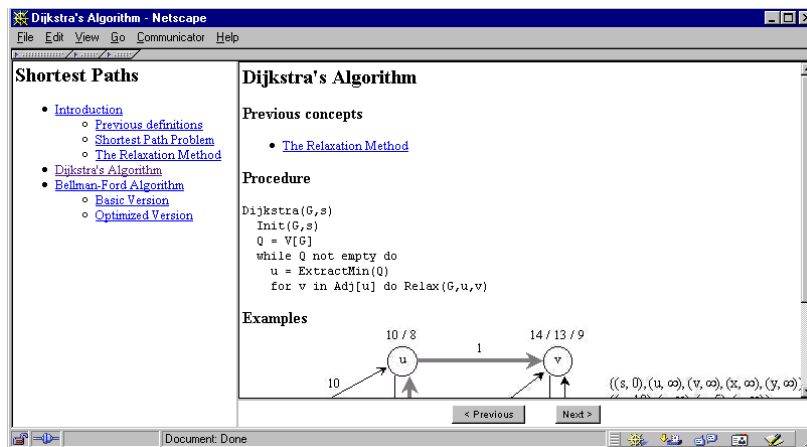


Fig. 1. Generated web page for a topic of type *Algorithm*

The template definition language supports the introduction of adaptive elements by using conditionals. For instance, in lines 4 to 7 in the preceding example, the presented information could be conditioned to the student's level of expertise, including all available examples when the student is a beginner, and a single example for more advanced students, showing the proof of correction only if it is relevant and not too difficult for the student:

```

<% if (user.expertise < 0.5) { %> <%= examples %> <% } %>
<% else { %> <%= examples.upto(1) %> <% } %>
<% if (correction.relevant && correction.difficulty < user.expertise) { %>
<h3> Proof of Correction </h3> <%= correction %> <% } %>

```

The expression language for templates includes other facilities that allow, for instance, cutting down, filtering or sorting lists according to an arbitrary comparison function, generating trees and linked lists by traversing a relation, or forcing the generation of hypermedia links. The basic template language allows the specification of a wide set of non-trivial presentations by using a very simple syntax. However the designer can write arbitrarily complex Java code inside the templates themselves.

4.2 Presentation Rules

Presentation rules govern aspects like link generation, correspondence between link styles and topic states, ordering and layout of (fragment or link) lists, and the generation of built-in presentations for topic network subsets like linked lists and trees. When, like in the previous subsection, the designer refers to a relation like *prerequisite* in the template for class *Algorithm*, rules take

care automatically of deciding whether to insert prerequisite details in the generated page, or to generate a link for each prerequisite, which style and annotation are used in the latter case, and how all the pieces are laid out. In doing so, the system analyzes whether the relation is simple or multivalued, the class of the involved topics or fragments, their state, and other conditions, if any, stated by the designer. The designer can modify existing rules and define her/his own.

Rules consist of a list of zero or more conditions, followed by the presentation to be applied when the conditions hold, described with the same syntax as used in templates. For example, the following rule establishes a green tonality color for links to knowledge items that have been read and whose prerequisites are all known by the student:

```
<Rule class="KnowledgeUnit">
  <test condition="asLink && read"/>
  <test-every var="item" list="prerequisites" condition="item.known"/>
  <presentation> <font color="#006600"> <%= this %> </font> </presentation>
</Rule>
```

(8)

To use bulleted HTML lists each time lists of links are to be displayed, a rule like the following can be defined:

```
<Rule class="List[KnowledgeUnit]">
  <test-every var="item" list="this" condition="item.asLink"/>
  <presentation>
    <ul> <iterate var="item" list="this"> <li> <%= item %> </li> </iterate> </ul>
  </presentation>
</Rule>
```

(9)

While writing rules is a delicate task that requires a familiarity with the system, any author with basic HTML knowledge could modify a rule like the one above to use, for instance, HTML tables instead of lists.

When the presentation systems receives a request to present an item, before applying the corresponding template PEGASUS tries to fire all applicable rules. When references to other objects appear in the right side of a rule (as part of a relation, or explicitly like in the expressions <%= this %> and <%= item %> in lines 8 and 9), these objects are processed in turn, applying rules again. When it is no longer possible to apply more rules, the template that corresponds to the object class is applied (in this sense, templates can be seen as lowest priority rules whose condition is *true*). This procedure is repeated recursively with all objects that appear in turn when processing the template.

5 Architecture

At runtime, the student interacts with the application from a web browser. The interaction with an application built with PEGASUS consists of traversing the domain object network. Each time the user moves to an object, PEGASUS responds by generating an HTML page. In doing so the system 1) resolves the user's request by determining the actual object to move to, 2) locates the instance in the domain model, 3) updates the domain and user models, 4) generates the HTML presentation applying the pertinent rules and the template that corresponds to the object class. In the generated pages links do not point to other pages but refer, explicitly or descriptively, to other domain objects.

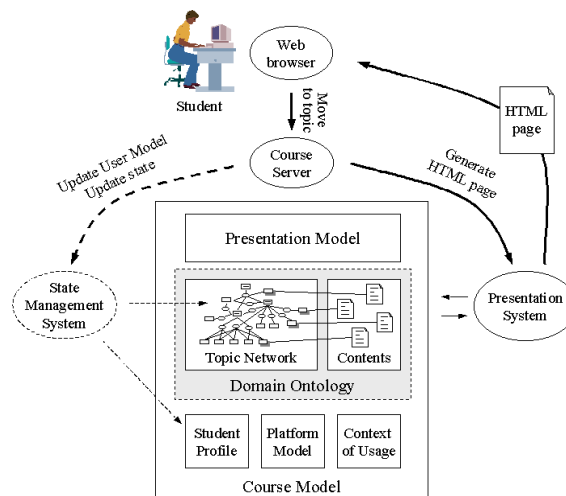


Fig. 2. Course model and system architecture

In most cases our presentation system will not work alone, since steps 1 and 3 above are external to PEGASUS. After an ontology for the subject matter is built, a runtime module is needed to set up and/or update topic networks (step 3), as illustrated in Figure 2. Optionally, a planner can be included like in DCG [13], to determine the path to follow in response to user's requests (step 1).

6 Conclusions

The proposed knowledge representation system can reproduce the domain models used in a wide range of hypermedia systems. Because the dynamic generation of presentation is a separated mechanism from the application state update mechanisms, PEGASUS is compatible with different courseware support tools like the ones described in the related work section. Our approach allows the specification of presentation independently from content construction, enhancing presentation reuse and consistency, thus reducing the development cost.

While the construction of templates is within reach of any web page designer who is familiar with HTML and JSP technologies, the definition of ontologies is a delicate task that requires the participation of an advanced designer, trained in using our system. Once the ontology and the associated presentation models are defined, the construction of the domain model is within reach of the average author. The introduction of modifications on presentation templates and rules can be an easy step for this kind of author towards a more advanced usage level.

PEGASUS has been implemented in Java™ (JDK 1.3), using XML/DOM and JavaServer Pages™ [10]. At the time of this writing we are about to complete a set of interactive authoring tools to facilitate the construction of ontologies and domain object networks. Among our future work plans we include the development of a graphical editing tool where authors can customize presentation models by example, by editing generated HTML pages, as in [4]. The creation of this kind of tool is not possible without an explicit declarative model of presentation.

7 Acknowledgements

The work reported in this paper is being partially supported by the Spanish Interdepartmental Commission of Science and Technology (CICYT), project number TEL1999-0181.

References

1. Brusilovsky, P., Eklund, J., Schwarz, E.: Web-based Education for all: a Tool for the Development of Adaptive Courseware. *Computer Networks and ISDN Systems*, 30, 1-7, 1998.
2. Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. In: Brusilovsky, P., Kobsa, A., Vassileva, J. (eds.): *Adaptive Hypertext and Hypermedia*. Kluwer Academic Publishers, 1998, 1-43.
3. Carro, R. M., Pulido, E., Rodríguez, P.: Dynamic generation of adaptive Internet-based courses. *Journal of Network and Computer Applications* 22, 1999, 249-257.
4. Castells, P., Szekely, P.: Presentation Models by Example. In: Duke, D.J., Puerta A. (eds.): *Design, Specification and Verification of Interactive Systems '99*. Springer-Verlag, Viena, 1999, pp. 100-116.
5. De Bra, P., Calvi, L.: AHA! An open Adaptive Hypermedia Architecture. *The New Review of Hypermedia and Multimedia*, 4. Taylor Graham Publishers, 1998, 115-139.
6. Macías, J. A., Castells, P.: A Generic Presentation Modeling System for Adaptive Web-based Instructional Applications. To appear in proceedings of ACM Conference on Human Factors in Computing Systems (CHI'2001), Extended Abstracts. Seattle, Washington, 2001.
7. Macías, J. A., Castells, P.: Adaptive Hypermedia Presentation Modeling for Domain Ontologies. To appear in proceedings of 10th International Conference on Human-Computer Interaction (HCI'2001). New Orleans, Louisiana, 2001.
8. Murray, T.: Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design. *Journal of the Learning Sciences* 7, 1, 1998, 5-64.
9. Pérez, T. A., Gutiérrez, J., Lopistéguy, P.: An Adaptive Hypermedia System. *Proceedings of Artificial Intelligence in Education (AIED'95)*. AACE, Charlottesville, 1995.
10. Sun Microsystems, Inc.: JavaServer Pages™ Technology. <http://java.sun.com/products/jsp>.
11. Trigg, R. H., Weiser, M.: Textnet: A Network-based Approach to Text Handling. *ACM Transactions on Office Information Systems (TOIS)* 4, 1, January 1986, 1-23.
12. Vassileva, J.: Dynamic Courseware Generation: at the Cross Point of CAL, ITS and Authoring. *Proceedings of International Conference on Computers in Education (ICCE'95)*. Singapore, 1995, 290-297.
13. Vassileva, J.: Dynamic Course Generation on the WWW. *Proceedings of 8th World Conference on Artificial Intelligence in Education (AIED'97)*. Kobe, Japón, 1997, 498-505.
14. Wilkinson, R., Smeaton, A. F.: Automatic Link Generation. *ACM Computing Surveys*, 31, 4es, December 1999.