GALE: A Highly Extensible Adaptive Hypermedia Engine

David Smits, Paul De Bra
Information Systems Group
Department of Computer Science
Eindhoven University of Technology
d.smits@tue.nl, debra@win.tue.nl

ABSTRACT

This paper presents GALE, the GRAPPLE Adaptive Learning Environment, which (contrary to what the word suggests) is a truly generic and general purpose adaptive hypermedia engine. Five years have passed since "The Design of AHA!" [4] was published at ACM Hypertext (2006). GALE takes the notion of *general-purpose* a whole lot further. We solve shortcomings of existing adaptive systems in terms of *genericity*, *extensibility* and *usability* and show how GALE improves on the state of the art in all these aspects.
We illustrate different authoring styles for GALE, including the use of *template* pages, and show how adaptation can be defined in a completely decentralized way by using the *open corpus adaptation* facility of GALE.
GALE has been used in a number of adaptive hypermedia workshops and assignments to test whether authors can actually make use of the extensive functionality that GALE offers. Adaptation has been added to wiki sites, existing material e.g. from w3schools, and of course also to locally authored hypertext. Soon GALE will be used in cross-course adaptation at the TU/e in a pilot project to improve the success rate of university students.

Categories and Subject Descriptors

H.5.4 [**Information Interfaces And Presentation**]: Hypertext/Hypermedia – *Architectures, Theory.*

General Terms

Experimentation, Human Factors, Languages.

Keywords

Authoring, adaptive hypermedia, adaptation engine.

1. INTRODUCTION

In Vannevar Bush' article "As We May Think" [8] the concept of *linking* information items was introduced. In a way the "Memex" device Bush envisioned was a form of *hypertext*. In addition, the user action of building "*trails of his interest through the maze of materials available to him*" was a first clear sign of using *personalization* to cope with information overload. Furthermore the personalization also included adding some kind of annotation: "*he inserts a page of longhand analysis of his own*". This shows that just linking information items (possibly from different authors) may not constitute a coherent story, hence the annotations or what we would call *content adaptation*.

The personalization envisioned by Bush was aimed at *revisiting* information (finding it again through trails), and at *recalling* a previously discovered meaning. When Bush defined *trailblazing* as a possible new profession we can understand this as doing personalization for others. Adaptive hypermedia research, first summarized by Brusilovsky in 1996 [5] and updated in 2001 [6] aims at automating this "trailblazing" through *link adaptation* and the annotations through *content adaptation*. Knutov et al [12] describe (in 2009) many new adaptation techniques developed to date and provide a list of challenges for creating a new *generic* adaptive hypermedia system, capable of dealing with *ontologies*, *open corpus adaptation*, *group adaptation*, *information retrieval and data mining*, *higher order adaptation*, *context awareness* and *multimedia adaptation*.

This paper aims to show how GALE (which to some extent is a successor to AHA! [2, 4]) tackles some of these challenges. It not only describes the core functionality of GALE but specifically shows (through examples and results of authoring experiments) how GALE can serve as the adaptation engine for different adaptive hypermedia applications.

This paper is organized as follows. Section 2 presents the global architecture of GALE. We concentrate on three aspects: the processor pipeline (for selecting and adapting resources), the configuration through which you can completely alter GALE's behavior, and the event bus and services through which you can make GALE compatible with different adaptation languages. Section 3 presents an application developed by a student to show the complementarity of the conceptual structure and the content of an adaptive application. We also list some other adaptive sites that were created by groups of students, with or without the graphical CAM authoring tool [10] developed in the GRAPPLE project. You can read Sections 2 and 3 in any order. Section 2 is a technical explanation and Section 3 contains examples. If this paper were adaptive (like [4]) this would have been more obvious. Section 4 describes the next big step in adaptive hypermedia authoring: *open corpus* adaptation, where content and adaptation are not only fused but can be distributed over arbitrarily many websites, with each page defining its own adaptation. Finally we discuss related work, some of which suggests future developments in adaptive hypermedia as we realize that no matter how generic GALE is a new and more generic generation of adaptation engines is expected to be developed in another few years time.

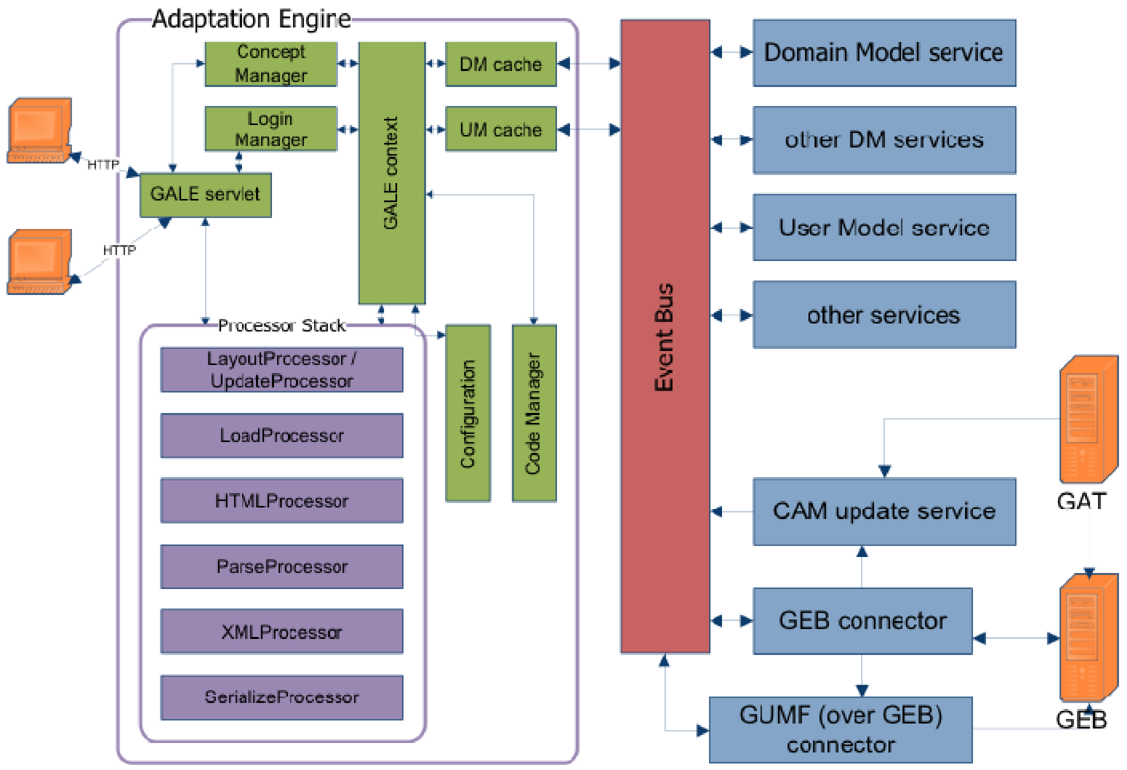


Figure 1. The architecture of GALE.

2. THE ARCHITECTURE OF GALE

Figure 1 shows the GALE architecture. We will concentrate on the purple part (bottom left) which deals with the adaptation to a resource (file or page), the green part (top left) which mainly deals with configuration, login and handling user requests, and some of the blue part (top part of the right side) which deals with internal and external services to handle adaptation formats and user modeling services. If you are reading this paper sequentially and you feel this section may be too technical to start with you should read section 3 first and then come back here. (**We mean it!**)

Let us first consider the overall process of GALE handling a request from a user (which typically comes in as a request for a URL, sent from the browser through http). We can visually represent this process as shown in Figure 2 (on the next page). The process involves interaction with the Domain Model (DM) of an application (describing the topic domain and adaptation) and the User Model (UM) which holds all the information GALE knows about each individual user.

1. If this is the first request the user sends since starting the browser no session will be associated with that request, so a session is initiated and a login procedure started. This is standard behavior for Web-based applications. GALE can work with several login managers (see Section 2.2) but for stand-alone GALE use the login follows the following multi-step procedure:
   a. For a first request (without session information as there is no session yet) the user is still unknown. The login manager redirects to a servlet/page that prompts the user for a user id and password.
   b. The user id is passed on to the UM cache, to request the application-independent part of UM for this user. Internally GALE refers to this as the *user entity*.
   c. Since the UM cache will not have cached the user model yet, it will communicate with the user model service through the event bus.
   d. UM is needed by the login manager to verify that the user has provided the correct password. Next the login manager (servlet) returns a redirect to the original URL. As a result the user's browser will request the same concept again, this time with session information.
2. GaleServlet now calls the concept manager in order to find out how to handle the request. If the request is for a concept, the concept manager will determine the identity of the requested concept and retrieve the domain and adaptation data for the concept from the DM cache (which may need to load it from a Domain Model service). If the URL does not refer to a concept it is handled differently (e.g. a file can be simply retrieved and served, as in the case of an image).
3. Handling the concept is a multi-step sub-process that uses processors (more or less in the top to bottom order as shown in the purple part of Figure 1). GALE can be extended with new processors that can be used anywhere in the processing pipeline. (We describe these extensions in Section 2.2.) Section 2.1 describes the default processor pipeline.

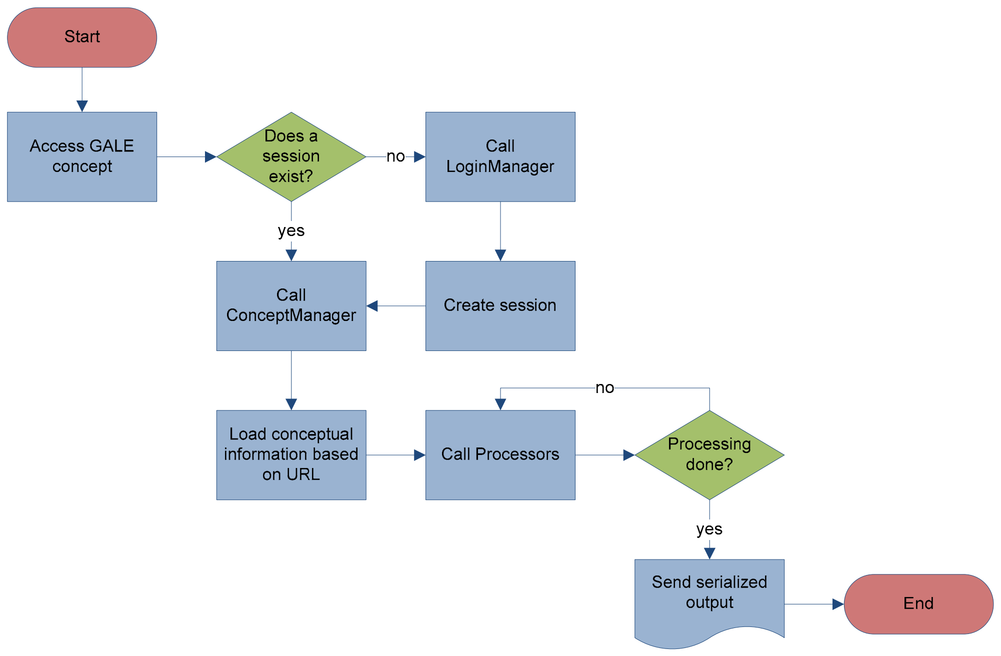


Figure 2. Handling a request for a concept.

2.1 The GALE Processor Pipeline

The processing of a concept (request) is done through a series of processors (each one handling the output of the previous one). The processors are controlled using a *level* that is updated each time a processor finishes. The association of acceptable levels is done in the configuration (see Section 2.2), thus defining a very simple process model. One can *insert* a processor into the processor chain by choosing the range of levels in which the processor becomes active.

GALE uses its own language GAM to define an application's domain and adaptation model (DM). Section 4 contains some of that code. Here we use a subset of GAM which we refer to as *GALE code*. This is code to either evaluate an expression over information from DM and UM or to update values in UM. UM is the main information source for the personalization (or adaptation) offered by GALE. We sometimes use a *property* of a concept, which is a DM element, and sometimes an *attribute*, which is a UM element. The GALE code is in fact Java code in which some shorthand notation is used to refer to concepts, properties and attributes. Section 3 gives several examples of GALE code. Here we just give a bare minimum introduction:

${#suitability} refers to the *suitability* attribute of the current concept (# always refers to a attribute).

${#image?title} refers to the *title* property of the *image* attribute of the current concept (? always refers to a property).

${->(parent)?type} refers to the *type* property of the *parent* of the current concept (following the parent relation).

#{#visited, ${#visited}+1}; is a statement that assigns the result of ${visited}+1 to the *visited* attribute of the current concept. It thus simply increments the *visited* attribute by 1.

We now explain the processors of the pipeline in the order they are called (which is the top to bottom order in Figure 1).

1. The first processor that is called is the *UpdateProcessor*. It signals an *EventManager* that the 'access concept' event has occurred. The default *EventAccessHandler* executes the *event code* of the concept as defined in DM. Typically this event code will signal some UM update to the UM cache. For instance, the *counter of visits* to the concept may be incremented. UM cache communicates over the event bus with the UM service. *Event code* associated with UM attributes may prompt the UM service to generate more UM updates. The resulting changes to UM are posted on the event bus, and as a result are integrated in the UM cache. So, important to note here is that 1) UM updates are calculated before generating adaptation (a design choice that corresponds to previous behavior of AHA! and motivated and explained in [4]), and 2) UM updates come from both the *UpdateProcessor* (in the adaptation engine) and from (event code) rules executed within the UM service. This distributed execution of UM updates is essential in GALE as GALE can also be used together with external UM services, including the GRAPPLE user Model Framework GUMF [1].

2. After the UM updates have been performed the *LoadProcessor* will retrieve the actual resource (file) associated with the concept. The name of the resource is found in the *resource attribute* of the concept. This "name" may refer to a local file, a file to be retrieved from some server using http, or may be a (GALE code) expression over DM and UM to "compute" the name of the actual resource. An *InputStream* is opened so that a subsequent processor can load and process the data. File name extensions are used to determine the mime type of the resource.

3. Optionally the *LogProcessor* then adds an entry to a global log file (access.log by default). The id of the user, date, request, referrer (that may be present in the HTTP request),

the name of the requested concept and the resulting resource are logged, for possible later analysis. Note that as the *UpdateProcessor* runs before the *LoadProcessor* it cannot log the name of the resource somewhere in the user model as that name is not yet known at the time. But the resource name can be logged by the *LogProcessor* which runs later.

4. If the mime type of the resource is some kind of HTML (but not XHTML) the HTMLProcessor uses the (open source) Tagsoup[1] converter to convert the file to XHTML. The new InputStream now contains valid XHTML.
5. If the input is XML (also XHTML) the *ParseProcessor* converts the input into an in-memory DOM tree, using the open source dom4j[2] parser. (We will not describe the processing of non-XML input further.)
6. The *XMLProcessor* walks through the DOM tree in order to perform adaptation where needed. The *modules* that may be used to perform adaptation to certain tags are loaded by the XMLProcessor. The configuration file (see Section 2.2) indicates which XML tag is handled by which module. Modules are provided to handle "if" tags, "object" tags, links, variables, and more. Adding new modules to handle different tags, possibly in different XML formats, is relatively easy. Within the GRAPPLE project the addition of modules has been investigated for *device adaptation* and for *adaptation to virtual reality* [16].
7. Optionally, the *LayoutProcessor* generates a frame-like structure using tables, by creating an (in-memory) XML document that contains the *views* (any class that implements the *LayoutView* interface) embedded in a table that defines the layout. This XML document has a placeholder element where the actual content should be. A *CSSLayoutProcessor* is available that uses *css* and *div* sections to layout the browser screen. A *FrameLayoutProcessor* can also be used that uses an *iframe* element to display the actual content. The different processor choices make it possible to combine GALE with many other presentation structures (e.g. with portals or learning management systems).
8. When the DOM tree is adapted the *SerializeProcessor* generates the textual XML representation and presents that to *GaleServlet* as an *InputStream*. For resource types that do not have specific processors associated with them GaleServlet will create this InputStream itself in order to send the content back to the browser. This happens for instance with images embedded in HTML pages. (For some special resource types GaleServlet calls a special PlugIn that may generate its own output. These plug-ins set the level to 100, which for GaleServlet means that the output was already generated by the plugin itself. Examples of such plug-ins are the Password and the Logout PlugIn.)

2.2 GALE Flexibility through Configuration

GALE was developed as part of the GRAPPLE EU project. GRAPPLE aimed at integrating open source or commercial Learning Management Systems (LMS) with Adaptive Learning Environments (ALE). The project aimed at making adaptivity available to a broad audience (mainly in technology-enhanced learning). To this end the adaptation engine needed to be made very generic and extensible, because developers of adaptive applications are expected to have special desires for adaptation functionality we might not foresee. Within the project itself *adaptation in simulation* and *adaptation in virtual reality* were already considered, as well as *device adaptation*. Different LMS may require a different way of *embedding* GALE output in their presentation. The different layout processors make this possible. To make GRAPPLE usable with many different LMS and to support *life-long learning* a repository of user-specific information was needed (the GRAPPLE User Modeling Framework, GUMF [1]). Different LMS store information in GUMF, and Section 2.1 already explained how input from such an external UM service is captured by GALE's UM cache. The integration process between LMS and ALE is described more in detail for instance in [15].

GALE uses the *Spring* "inversion of control"[3] container to configure and instantiate all components. Without going too much in detail we describe the most important configuration elements here (omitting small things like where the access log file or some other files are stored or at which address GEB and GUMF are located). The GALE configuration is stored in the file *galeconfig.xml* (in Tomcat's webapps/gale/WEB-INF directory). What the configuration mostly does is associate names that have meaning as functional parts of GALE with a Java class name (implementing the functionality) and it also defines which properties configure the behavior of that functional part.

1. The *processorList* defines processors that handle a request and adapt resources. The main processors have been mentioned in Section 2.1 already. Here we look specifically at two processors: *XMLProcessor* and *PluginProcessor*.
   a. The *XMLProcessor* performs transformations (for adaptation) to the DOM tree of an XML resource. The configuration contains a list of *Modules* that handle specific XML tags. Adding adaptation to a new tag can be done by creating a new module and associating it with the tag in this list. We only present a small selection from the default modules:
      - The *IfModule* handles the <if> tag. It expects <if> to have an argument "expr" that is a Boolean expression in GALE code. It expects one or two child elements: a <then> and optionally an <else> element. The module replaces the <if> subtree by either the content of the <then> subtree or the <else> subtree. The IfModule thus realizes what is known as the *adaptive inclusion of fragments* technique [12].
      - The *AdaptLinkModule* handles the <a> tag which is used just like the HTML <a> tag, but referring to a *concept*, not a page or resource. GALE (actually the *LoadProcessor*) decides which page to retrieve and return based on the *resource* attribute of the concept. An optional *exec* argument can be used to associate a (UM) action with following the link. For instance: *exec="#{tour#start,true};"* could be used to say that following *this* link indicates the start of a tour, whereas accessing the same concept (tour) through *other* links does not.
      - The *ObjectModule* inserts either a file specified through the *data* argument or a concept specified

[1] See ccil.org/~cowan/XML/tagsoup/ for more information.

[2] See dom4j.sourceforge.net for more information on dom4j.

[3] See www.springsource.org for more details.

through the *name* argument. The <object> tag is preferred over the <if> tag when the same fragment needs to be conditionally included in many different pages. With *data="header.xhtml"* we can insert a header file in a page, whereas with *name="programming"* we insert whichever resource is associated with the concept *programming* (possibly involving evaluating expressions to select a resource).

- The *VariableModule* inserts either the value of a UM attribute or the result of a GALE expression in the page. <variable name="#visited"/> for instance shows the number of visits to the current concept; <variable expr="${#visited}"/> does the same, but now as expression.
- The *AttrVariableModule* is similar but inserts its result in the surrounding element's tag.
  ```
  <img><attr-variable name="src"
          expr="${?image}"></img>
  ```
  uses the value of the *image* property of the current concept as the source (url) of an image to insert. Note that because of syntax restrictions of the XML language we could not use <variable> inside the <img> tag itself. (Arguments of an XML element cannot contain XML elements.)
- The *ForModule* repeats a fragment of XML for elements in a list.
  ```
  <for var="concept" expr="${<-(parent)}">
   <variable expr="${%concept?title}"/><br/></for>
  ```
  inserts a list of titles of the children of the current concept[4].
- The *PluginModule* generates a link to a plug-in. (It does not perform the plug-in code itself as this is done by the *PluginProcessor* described below. <plugin name="logout">Logout</plugin> results in a link through which the user logs out. Note that because GALE transforms <plugin> into a link anchor the description of the plugin (the word "Logout" in the example) must not contain an <a> tag but can contain other HTML tags if desired.

b. The *PluginProcessor* handles plug-ins that perform some function and then generate "complete" output. Examples of plug-ins are the *password* and *logout* plug-ins (to change the user's password and to log out), the *mc* plug-in to evaluate a multiple-choice test, *exec* to execute some GALE code and show the result (used mostly for debugging purposes), and *export* to generate a textual representation of a whole application (DM).

2. The *hibernateDataSource* bean controls how GALE stores data (including DM and UM). As we use Hibernate[5] GALE is independent of the database backend used. We have used two storage methods so far: hsqldb which stores data in a simple text format and mysqldb to store data in a MySQL database. MySQL (or any other real database like Postgres or Oracle) is recommended for a real server installation. In that case it is possible to use a separate machine for the database to offload the GALE server.
3. Figure 1 shows that GALE uses an internal *event bus* through which the core GALE engine communicates with DM and UM and other services. Two implementations of this bus exist: one (*LocalFactory*) that uses method calls and thus requires all services to reside on the same server and one (*SOAPFactory*) that uses SOAP and can handle DM and UM services that run on different machines. The event bus is configured to communicate with a number of services. (This bus uses the Publish/Subscribe method.) The DM and UM services are most obviously needed, but several other services exist to implement compatibility of GALE with different authoring formats and tools for adaptive application, as described in Section 2.3 below.
4. The *loginManager* bean defines how GALE users can identify themselves. The *DefaultLoginManager* presents a simple form for username and password (explained in Section 2.1). The *LinkLoginManager* allows LMS users to automatically log in on GALE using the id they have on the LMS. The LMS uses a "secret" key to identify itself to GALE and essentially "promises" that the user is who (s)he says (s)he is. The IdPLoginManager makes use of Shibboleth[6] to make GALE usable in a federation of institutes/companies that share a single sign-on facility.
5. The *codeManager* defines which language of adaptation code is used. All examples of GALE code used in this paper use GAM: Java with some shorthand to refer to DM and UM values, but it is possible to use very different code provided that a new code manager is developed. Note that in GALE there are actually two code managers (slightly different): one in the adaptation engine (dealing with concept event code and GALE expressions used in GALE XML tags) and one in the UM service (dealing with UM attribute event code).
6. The *configManager* is a wrapper for handlers of different types of configuration: for *processors*, for *link adaptation* (with icons) and for *presentation*. The latter (*Presentation-Config*) defines which automatically generated parts can exist in the presentation of adapted pages. By default there is a *static-tree-view* which displays a menu over the hierarchy of concepts of the current application (see Figure 6), *next-view* which generates a link to the next concept in a guided tour (see Figure 5), and *file-view* which inserts (and adapts) a file with a fixed name. The file-view is used to include a header and/or footer for instance (see Figures 5 and 6).

2.3 GALE and other Adaptation Formats

Fifteen years after the first seminal adaptive hypermedia paper [5] it is an illusion that a new *general purpose* adaptation engine can be designed and built that ignores all earlier attempts at defining and implementing languages for specifying adaptive hypermedia applications. As Figure 1 already shows GALE's event bus can handle different "Domain Services". GALE stores "Domain Models" (including adaptation rules) using its built-in DM service based on GAM (for examples see Section 4). A new GALE application is normally developed using GRAPPLE's CAM editor [10] (also called Course editor) and then *compiled* into a GALE domain model. Using the *export* plug-in such a model can be represented in a textual format, called GDOM. When you edit

[4] The construct <- in fact has to be written as &lt;- because of an XML syntax restriction, but we wrote <- for clarity. We make this "error" throughout the paper to improve readability.

[5] See www.hibernate.org for more information.

[6] See shibboleth.internet2.edu for more details.

such a GDOM file and save it (in GALE's "config" directory) the GDOM domain service will load and interpret it. (GDOM was used as GALE input format while the GRAPPLE authoring tools were still being developed and has now been superseded by GAM.) Likewise you can store an AHA! version 3 ".aha" file in GALE's "config" directory and the AHA3 domain service will load and interpret it. Existing AHA! applications can thus easily be ported to GALE. The course "Hypermedia Structures and Systems" (that has been running at the TU/e since 1993 and that was made adaptive in 1996) has not only been running on all versions of AHA! but is currently being served by GALE. Other AHA! applications will soon follow (including the AHA! Design paper [4]) as we are phasing out our existing AHA! installations.

Creating domain services for different formats is easy as long as these formats are based on the idea of "concepts" (with some associated data structure) and "adaptation rules" that can be translated to the model with *event-condition-action rules* used by GALE and as long as content (pages) is *written* by the author (not generated completely from a Web-based information system). Well-known systems like Interbook [7] and KBS-Hyperbook [11] follow this model and their applications can be "easily" imported by adding a GALE DM service. (In fact, Interbook was already translated to AHA! version 3. [3]). Dedicated DM services for alternative formats handle differences in syntax but cannot deal with very different conceptual application models.

Languages like LAG [9] may require some content generation and are thus more difficult to convert. (GALE can only do limited content "generation" through the <for> tag and *ForModule* explained in Section 2.2.) When content needs to be generated a DM service alone is no longer sufficient, but a compiler can generate the GALE DM and content. This has been done for LAG to AHA! already. Once such a compiler is involved it can generate a "real" GALE DM, alleviating the need for a special DM service.

Within GRAPPLE the language GAL was developed [14] for which GALE does not (yet) have a domain service. GAL allows the use of arbitrary query languages to express queries over the domain and user model. In order to implement GAL first of all a concrete query language needs to be chosen. In [14] all examples use SPARQL. Translating SPARQL to GALE code would be difficult (although theoretically not impossible), but replacing GALE code by SPARQL could be done as GALE can be configured to use any code language (as long as an interpreter for the desired code language is added to GALE). Like with LAG the use of GAL also implies the generation of content, so a compiler rather than just a DM service would be needed.

3. APPLICATIONS REALIZED IN GALE

This section is an example-based introduction to (and motivation for) GALE. It can be read without first reading Section 2. This is our "poor man's" attempt to do personalization on paper.

We will not describe how to define an overall *adaptation strategy* or how to design the adaptation based on *pedagogical relationship types* such as *prerequisites*. This is the subject of GRAPPLE's (CAM) authoring too described in [10].

GALE supports two approaches towards creating the *content* of an application, represented in figures 3 and 4 (taken from [13]).

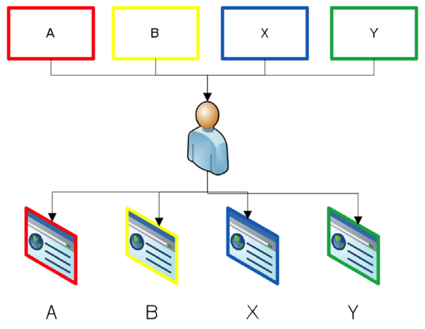


Figure 3. Creating pages separately.

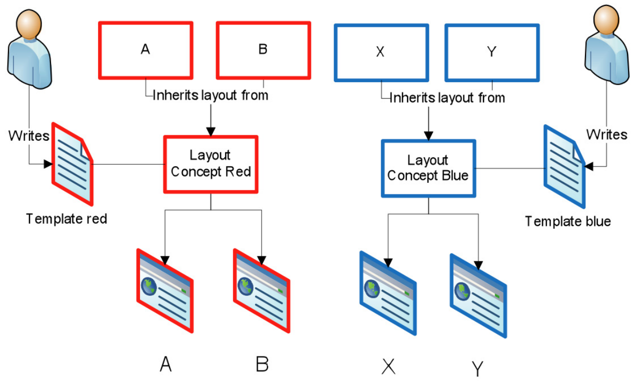
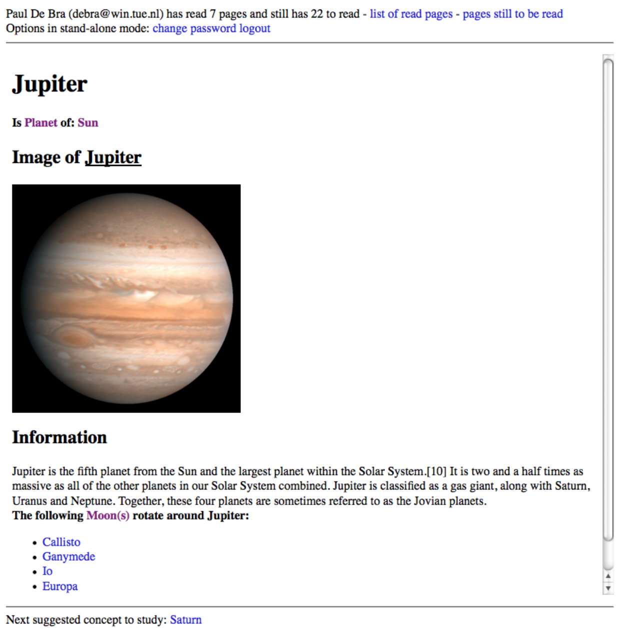


Figure 4. Authoring through template pages.

Figure 3 shows an author writing complete pages. This is a viable option for authors who wish to create adaptive applications without writing (or even seeing) any GALE code. This approach is also well suited for an application in which pages do not have a common structure. The hypermedia course at the TU/e was created in this way and the GRAPPLE tutorial (at http://gale.win.tue.nl/) as well. This is also a good approach for making existing applications adaptive, for instance applications generated from databases (possibly from a content management system or a wiki).

Figure 4 shows an example where a cluster of pages share the same structure. (A and B are alike, X and Y are also alike.) When pages are created separately one needs to be careful to replicate the style on all pages that should be similar, and changing the style involves changing all these pages individually. Therefore Figure 4 shows the use of one *template* page for each set of pages that should look alike. The templates are used to (virtually) create individual pages by indicating which bits to use where in the template. Changing the presentation of all pages that use the same template requires a single file to be edited. Figure 5 shows the presentation of a concept from the application *Milkyway* that was created by a master student at the TU/e [13]. All pages of Milkyway have the same look and feel, not only with the *header* and *footer* (and a navigation menu that we cut off) but also within the main content part of the page. Milkyway deals with stars, planets and moons, and has slightly different templates for each of these types of celestial bodies. Below we explain the template used for presenting planets.


Paul De Bra (debra@win.tue.nl) has read 7 pages and still has 22 to read - list of read pages - pages still to be read
Options in stand-alone mode: change password logout

Jupiter

Is Planet of: Sun

Image of Jupiter

Information

Jupiter is the fifth planet from the Sun and the largest planet within the Solar System.[10] It is two and a half times as massive as all of the other planets in our Solar System combined. Jupiter is classified as a gas giant, along with Saturn, Uranus and Neptune. Together, these four planets are sometimes referred to as the Jovian planets.
The following Moon(s) rotate around Jupiter:

- Callisto
- Ganymede
- Io
- Europa

Next suggested concept to study: Saturn

Figure 5. Example of a page based on a template.

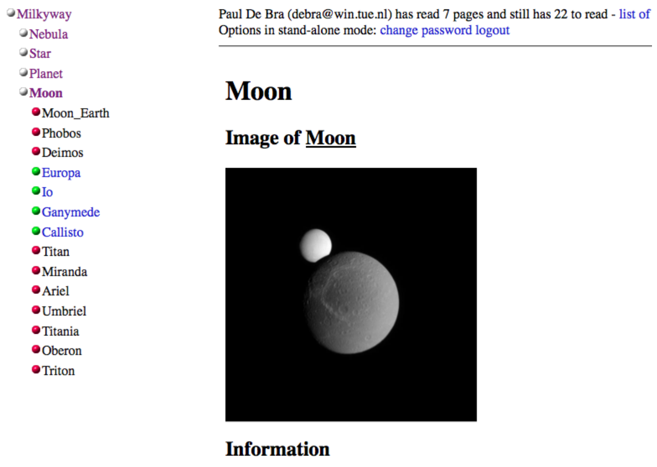
In a GALE application every concept can have an associated *layout*. In this example the header and footer are defined as part of that layout. The layout can be different for each concept and can also be adaptively changed but we do not expect this to be common[7]. The "main" part of the presentation is the *page*. In Milkyway the page consists of a *title* (Jupiter), a *typology* (Is Planet of: Sun), an *image* with *title*, an *information* fragment and a *list of links to children* (The following Moon(s) rotate around Jupiter.) Below we show how each part is defined, to give you a feel for creating pages containing GALE tags and GALE code. All domain model properties that are used (like "Jupiter", "Sun") and relations (like "parent" and "isPlanetOf") are created using the graphical GRAPPLE authoring tools (the Domain editor to be exact) that are not described in this paper.

- In the header we see the name and email of the user. The code <variable name="gale://gale.tue.nl/personal#name"/> and <variable name="gale://gale.tue.nl/personal#email"/> extracts that information from the user model and inserts it in the page.
- The title "Jupiter" is generated by means of the code <variable expr="${?title}"/> which inserts the *title* property of the current concept in the page. As shown in Section 2.1 the # sign refers to a user model attribute and the ? sign refers to a domain model property.
- The "Is Planet of: Sun" is a bit more complex. "Planet" is actually the *title* of the *parent* concept and "Sun" is the *title* of the concept to which Jupiter has an *isPlanetOf* relation.
  <a><attr-variable name="href" expr="${->(parent)?title}"/> generates the link anchor tag. Calculating the value for the "href" argument cannot be done inside the <a> tag so it is done using <attr-variable> (see Section 2.1). The anchor text is <variable expr="${->(parent)?title}"/> (which generates the word "Planet"). The link to "Sun" uses <a><attr-variable name="href" expr="${->(isPlanetOf)?title}"/> <variable expr="${->(isPlanetOf)?title}"/></a>. Like with "Planet" the same expression appears twice because "Sun" is the anchor text as well as the link destination.
- To insert the image (we ignore the title here) we use the code <img><attr-variable name="src" expr="${?image}"/></img> where the name (URL) of the image is part of the domain model (created with the Domain editor). As you see the attr-variable tag can be used in combination with any other tag.
- The information fragment is stored in a separate file and included in almost the same way as the image: <object><attr-variable name="data" expr="${?info}"/></object>. The name of the file is part of the domain model.
- The list of moons is generated using the <for> tag. (Milkyway additionally checks whether a planet has moons to avoid generating an empty list if it doesn't.) We only show the code for the list (not for "The following moons rotate…")

```
<ul>
<for var="concept" expr="${<-(isMoonOf)}">
   <li><a><variable expr="${%concept?title}"/>
      <attr-variable name="href"
           expr= "&quot;%concept&quot;"/>
   </a></li>
</for>
</ul>
```

  All concepts with an *isMoonOf* relation to the current concept are associated, one by one, with the variable "concept". A bullet list is generated with for each such concept a list item with a link to the concept and with the title of the concept as link anchor text.

The Milkyway application has a "Course model" in which *prerequisites* are defined. For instance, a planet should be studied before studying its moons is recommended. Figure 6 shows the page about the concept "Moon". This time we have not cut off the navigation menu. In this menu we see link annotation (with link colors and with colored balls) to indicate that only the four moons of Jupiter are recommended (as we only studied the planet Jupiter and no other planet that has moons).



Paul De Bra (debra@win.tue.nl) has read 7 pages and still has 22 to read - list of
Options in stand-alone mode: change password logout

Moon

Image of Moon

Information

Figure 6. Link adaptation based on prerequisites.

The link annotation shown in Figure 6 uses the same "good", "neutral" and "bad" link classes (blue, purple, black) that were

[7] The hypermedia course actually does use adaptive layout to show or hide a navigation menu. The adaptive layout is the only observable difference between the course previously served through AHA! and the new version served by GALE.

previously used in some AHA! applications. In GALE (and in fact already in AHA! as well) you can define arbitrarily many link classes and arbitrary conditions for deciding which class a link should have. The default class and color scheme are suitable for the default adaptation (or pedagogical) models created using the GRAPPLE authoring tools (described in [10] for instance). The graphical authoring tools hide the complexity of the GALE adaptation rules from authors, but they do enable (advanced) authors to create their own arbitrarily complex rules (specified in the GRAPPLE Adaptation Model language GAM).

At the TU/e we had (groups of) students develop their own GALE applications, using any content they desired, and using mostly the GRAPPLE authoring tools for defining the adaptation. Some of the applications they developed are:

- basic Web tutorials (mostly about HTML and CSS) based on learning material from W3Schools;
- tutorials for programming languages or technologies (PHP, Python, C#, OpenGL);
- sports tutorials (tennis & tennis competition);
- an adaptive information site on Dutch universities;
- an introduction to the animal kingdom (not the Disney one);
- a tutorial on drinks (hot, cold, alcoholic or not);
- a Star Trek Voyager tutorial;
- an adaptive tutorial on hypertext.

All of these examples used adaptive navigation support through *prerequisite* relationships, some added content adaptation, some used template pages, some developed their own layout, some used content from a wiki, and one group even invented a new relationship type (the *disjunctive-prerequisite* which requires only one source concept to be known instead of all source concepts).

All students had comments (criticism) on the GRAPPLE authoring tools for creating the conceptual structure. Very few had any problems with (or gave negative comments about) the GALE adaptation engine. All students reported that once the authoring tools were more or less mastered the design and implementation of their adaptation models took between 10 and 15 hours (for applications of around 50 concepts and pages). This answers a commonly asked question: how much additional work there is in creating an adaptive versus a non-adaptive information website.

4. OPEN CORPUS ADAPTATION IN GALE

A serious limitation in adaptive systems to date is that the adaptation is designed (or at least stored) in one place and the adaptive application must be used on a single server. Using the GRAPPLE authoring tools concepts can be associated with resources (files/pages) from all over the Web, and using the GRAPPLE User Model Framework (GUMF) different adaptive applications can share user model information.

The goal of *open corpus adaptation* in GALE is to allow all relevant *conceptual* information (the GALE DM) to be stored outside the GALE server, anywhere on the Web. When this is done different applications on different GALE servers can use the same concepts. To reach this goal a new DM service has been developed. The DM for a whole application can be stored in a single file on any website, or the DM part for a single concept can be stored in a file that contains that concept and possibly also an associated page. Our illustration of the open corpus adaptation model in GALE shows some GAM code that is just like what the GRAPPLE authoring tools already produce. So it is only a small step to make these authoring tools produce the open corpus material instead of sending it to a central GALE server.

The open corpus DM service retrieves a DM concept and page according to the following scheme:

1. Load the resource specified by the URL. The resource is assumed to be HTML or XHTML.
2. Scan the page for the ‘gale.dm’ meta element (<meta name= ’gale.dm’ content=’… …’ />) and use its content in step (3).
3. If the content is a ‘redirect’ instruction (example: ‘redirect:elearning.xhtml’), use the specified relative URL to continue from step (1).
4. The content is interpreted as DM information stored in the GAM format. Multiple concepts may be described in the GAM code of a single file (even a complete application). Concept names are relative to the URL of the file containing the GAM.
5. If present, return the concept DM for the originally requested concept. If the concept DM does not already contain a resource attribute, it will be generated to point to the original resource specified by the URL.

The resource associated with the concept is then retrieved by the LoadProcessor in the usual way (looking at the resource attribute).

Below is an example http://gale.win.tue.nl/elearning.xhtml with the following content[8]:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns=http://www.w3.org/1999/xhtml
     xmlns:gale="http://gale.tue.nl/adaptation">
   <head>
     <meta name="gale.dm" content="
   { #[visited]:Integer `0` {
      event `if (${#suitability} && ${#read} < 100) #{#read, 100};
        else if (!${#suitability} && ${#read} < 35) #{#read, 35};`}
    #knowledge:Integer !`GaleUtil.avg(new Object[]
    {${<=(parent)#knowledge},${#read}}).intValue()`
    #[read]:Integer `0`
    #suitability:Boolean `true`
     event `#{#visited, ${#visited}+1};` } " />
   </head>
   <body>
   <p>This page is a placeholder for the elearning concept.</p>
   </body>
</html>
```

This concept has GAM code that states how the persistent attributes *visited* and *read* are updated and how the volatile attributes *knowledge* and *suitability* are calculated. Clearly authoring only becomes feasible if templates with such code are made available (so authors need not study GAM). Fortunately the example above can already be used as such a template and the templates can be offered on (and used from) any website to be used in adaptive applications on any other website. Another page can “inherit” this adaptation (GAM) code as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns=http://www.w3.org/1999/xhtml
     xmlns:gale="http://gale.tue.nl/adaptation">
   <head>
      <meta name="gale.dm" content=
```

[8] To improve readability we have used && instead of &amp;&amp: and < instead of &lt;

```
        "{->(extends) http://gale.win.tue.nl/elearning.xhtml}" />
  </head>
  <body>
  <p>This page uses the elearning template.</p>
  </body>
</html>
```

When a whole application domain is stored in a single file the “meta” element for the concepts/pages would look like:

```
<meta name='gale.dm' content='redirect:course.gam' />
```

and the file “course.gam” might have contents like:

```
welcome.xhtml {
  ->(extends)http://gale.win.tue.nl/elearning.xhtml
  ->(extends)layout.xhtml
  <-(parent)gale.xhtml
  <-(parent)gat.xhtml
}
gale.xhtml {
  ->(extends)welcome.xhtml
  ->(parent)welcome.xhtml
}
gat.xhtml {
  -> (extends)welcome.xhtml
  ->(parent)welcome.xhtml
}
layout.xhtml {
  #layout:String `
  <struct cols="250px;*">
    <view name="static-tree-view" />
    <struct rows="60px;*;40px">
      <view name="file-view" file="gale:/header.xhtml" />
      <content />
      <p><hr />Next suggested concept to study:
          <view name="next-view" /></p>
    </struct>
  </struct> `
}
```

We plan to soon offer the GRAPPLE authoring tools to generate the open corpus format so that authors can develop and publish adaptive hypermedia information of which the adaptive behavior can be used when served through different GALE servers (to different users).

5. DISCUSSION (AND RELATED WORK)

Transforming a document is one of GALE's core functions. Since GALE mostly works on HTML and XHTML content, an obvious question would be why GALE does not use XSLT to transform these documents. The approach GALE uses to adapt documents (using processors and in the case of XML, modules for each element) might seem unnecessarily complicated.

Furthermore, the idea of using Java code inside an HTML or XHTML page is also not new. Many techniques exist that allow part of a page to be dynamic, like JSP, Facelets[9], Velocity[10] and many more.

[9] See java.net/projects/facelets for more information.

[10] See velocity.apache.org for more information.

Before going into detail on the various options for processing XML and XHTML, consider that the notion of a processor as an object that transforms a resource into something more suitable for the user is more generic than XSLT, JSP, etc.. A processor can use the HttpServletRequest to obtain, modify and generate any form of output. The input and output of a processor can be as generic as the Java InputStream and OutputStream. Different processors can be chained to reuse functionality. GALE is not bound to XML to perform adaptation (or to any single format for that matter).

The XMLProcessor provides the main logic behind content adaptation in GALE. From a processing point of view, a better name might have been the DOMProcessor since it uses an in-memory DOM tree to perform adaptation (vs. SAX processing). A similar approach to processing can be found as early as AHA! version 2 (2002) [2]. At that time XML (1998), XHTML (2000) and XSLT (1999) were all fairly new technologies. The HTML handler in AHA! 2 used the Java Spring HTML parser and would perform adaptation on HTML independently of any XML or XHTML processing facility in AHA!. Building an in-memory DOM tree from an XHTML document was similar to the HTML-handler way of processing.

GALE has its roots in AHA!. Over the course of almost a decade the goal has always been to make an adaptation engine that is as generic (and extensible) as possible, while still maintaining simplicity, performance and ease of authoring. Where possible existing open source technologies have been used, like Tomcat, dom4j, Tagsoup, Spring, Hibernate, Shibboleth, etc. For adapting XML writing modules that adapt specific elements was found to be a very flexible approach. Adding adaptation to non-HTML formats such as X3D and SMIL is very straightforward.

As GALE is a fully modular and highly configurable system, we envision new ways of adapting information access in conjunction with GALE. With the current processors and modules for instance we cannot yet realize a real recommender system or an adaptive search facility, whether it be content-based or collaborative. The guided tour (using the “next” view) shows that adding new *views* to GALE are the first step towards search and recommendations.

People sometimes ask (as they did about AHA!) how GALE compares to some other adaptive hypermedia tool or application, and how good the adaptation is (for instance in terms of learning outcome when used for an adaptive course). These questions are about as meaningful as asking how good websites served by the Apache server are when compared to websites on for instance Microsoft Web Server. GALE is a *generic* and *general purpose* adaptation platform. Its adaptation is as good as what its users (authors) create. The strength of GALE compared to other adaptive hypermedia systems is that it aims to be able to emulate all of them (because the code used in GALE expressions and statements is arbitrary Java code). Current limitations in GALE are that 1) the code runs in a sandbox environment which prevents access to the full (Tomcat) server functionality (for security reasons) and that 2) we currently do not allow adaptation code for a user to access other users’ user models (for privacy reasons). GALE can handle *group adaptation* but this is currently not used so as to ensure that we respect all users’ privacy concerns. Defining a *group entity* (in addition to the *user entity*) is certainly possible. An entity can be associated with any number of other entities (and rules can update the other entity user models). Using this to implement collaborative filtering is still future work.

6. CONCLUSIONS

With the GALE adaptation engine we try to offer virtually all possible forms of adaptation for hypermedia applications (for now, based on individual user models). We have explained the architecture of GALE and stressed the genericity and extensibility. We hope to have struck the right balance between showing some concrete constructs and examples that can be realized in GALE and not suggesting that GALE might be limited to what we have shown.

We have deliberately not shown the authoring tools (described in [10]). These tools make a "default" type of adaptive application easy to design, but although a lot of GALE functionality can be tapped into through these tools, describing how to achieve this would be a complete paper on its own.

Adaptive hypermedia systems are typically limited to a single type of application and are also limited to "local" applications. GRAPPLE already allows the use of an adaptive application from within different Learning Management Systems, and sharing user model information through a common UM framework. But GALE adds the ability of using decentralized applications through *open corpus adaptation*. And this was added together with the ability to reuse adaptation templates so authors can use open corpus adaptation without needing to write (GAM) adaptation code.

The proof of the pudding is in the eating. In 2006 we explained AHA! version 3 through a paper that was itself adaptive [4]. For the GRAPPLE project (including GALE and the authoring tools) we also have an online adaptive description (tutorial) available, at http://gale.win.tue.nl/. We have not made this paper adaptive because an adaptive description of GALE would inevitably use only a small fraction of GALE's functionality, possibly suggesting that GALE applications should all be similar to that paper. Although GALE offers many adaptation possibilities we do not advocate using many adaptation possibilities in a single application. All reasonable and usable adaptive applications use adaptation *cautiously* and thus *sparingly* in order to avoid turning a usable information site into an unusable flashy adventure game. Our experiments with student-generated applications have (informally) shown that defining and implementing adaptation in a variety of adaptive applications is relatively straightforward and does not take an inordinate amount of effort and time.

Besides working on novel ideas for extensions (like search and recommendations) we plan on using GALE for the delivery of more adaptive courses, and specifically on including adaptation in courses based on user performance in other (earlier) courses, in order to improve the long term study performance of our bachelor and master students.

Acknowledgements

We wish to thank the European Commission, project 215434 (GRAPPLE) for their financial support.

References

[1] Abel, F., Henze., N., Herder, E., Krause, D., *Interweaving Public User Profiles on the Web*, In Proceedings of UMAP 2010, User Modeling Adaptation and Personalization, LNCS 6075, pp. 16-27, Springer, 2010.

[2] De Bra, P., Aerts, A., Smits, D., Stash, N., *AHA! Version 2.0, More Adaptation Flexibility for Authors*. Proceedings of the AACE ELearn'2002 conference, pp. 240-246, 2002.

[3] De Bra, P., Santic, T., Brusilovsky, P., *AHA! meets Interbook and more…,* Proceedings of the AACE ELearn 2003 Conference, pp. 57-64.`, 2003.

[4] De Bra, P., Smits, D., Stash, N., *The Design of AHA!*, Proceedings of the seventeenth ACM Conference on Hypertext and Hypermedia, pp. 133-134, 2006, with adaptive version at http://aha.win.tue.nl/ahadesign/.

[5] Brusilovsky, P., *Methods and techniques of adaptive hypermedia*, User Modeling and User Adapted Interaction, Vol. 6, pp 87-129, Kluwer Academic Publishers, 1996.

[6] Brusilovsky, P., *Adaptive Hypermedia*, User Modeling and User Adapted Interaction, Vol. 11, pp 87-110, Kluwer Academic Publishers, 2001.

[7] Brusilovsky, P., Eklund, J., Schwarz, E., *Web-based education for all: A tool for developing adaptive courseware.* Computer Networks and ISDN Systems (Proceedings of the 7th Int. World Wide Web Conference, 30 (1-7), pp. 291-300, 1998.

[8] Bush, V., *As We May Think*, The Atlantic Monthly, July 1945.

[9] Cristea, A.I., Smits, D., Bevan, J., Hendrix, M. *LAG 2.0: Refining a Reusable Adaptation Language and Improving on Its Authoring*, Proceedings of the 4th European Conference on Technology Enhanced Learning: Learning in the Synergy of Multiple Disciplines, Springer LNCS 5794, pp. 7-21, 2009.

[10] Hendrix, M., Cristea, A.I., *Design of the CAM model and authoring tool*. A3H: 7th International Workshop on Authoring of Adaptive and Adaptable Hypermedia Workshop, 4th European Conference on Technology-Enhanced Learning, 2009.

[11] Henze, N., *Adaptive hyperbooks: Adaptation for project-based learning resources*. PhD Dissertation, University of Hannover, 2000.

[12] Knutov, E., Bra, P.M.E. de, Pechenizkiy, M., *AH 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques.* New Review of Hypermedia and Multimedia, 15(1), 5-38, 2009.

[13] Ploum, E., *Authoring of adaptation in the GRAPPLE project*, Master Thesis, Eindhoven University of Technology. (available at http://alexandria.tue.nl/extra1/afstversl/wsk-i/ploum2009.pdf), 2009.

[14] Van der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.J., *GAL: A Generic Adaptation Language for describing Adaptive Hypermedia*, Proceeding of the International Workshop on Dynamic and Adaptive Hypertext: Generic Frameworks, Approaches and Techniques (DAH'09) in conjunction with ACM Hypertext 2009, July 2009.

[15] Van der Sluijs, K., Höver, K., *Integrating adaptive functionality in a LMS*, International Journal of Emerging Technologies in Learning (IJET), Vol. 4, nr. 4, pp. 46-50, 2009.

[16] De Troyer, O., Kleinermann, F., Pellens, B., Ewais, A., *Supporting Virtual Reality in an Adaptive Web-based Learning Environment, Learning in the Synergy of Multiple Disciplines*, In proceedings of the 4th European Conference on Technology Enhanced Learning (EC-TEL), LNCS 5794, pp. 627-632, Eds. Cress. U. et al, Publ. Springer, ISBN 978-3-642-04635-3, Nice, France, 2009.