A Fully Generic Approach for Realizing the Adaptive Web

Paul De Bra and David Smits

Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
debra@win.tue.nl, d.smits@tue.nl

Abstract. It is time for Adaptive Web (server) extensions to grow up and become generic. The GRAPPLE[1] (EU FP7) project aimed at integrating Learning Management Systems (LMS) with Adaptive Learning Environments (ALE) in order to support life-long learning. But instead of developing a dedicated Web-based ALE we developed an architecture containing a fully generic adaptive Web server, a distributed User Modeling Framework and a generic browser-based authoring environment for Domain Models and Conceptual Adaptation Models. The GRAPPLE architecture can be used for creating and serving any type of adaptive Web-based application. It supports content-, link- and presentation (layout) adaptation based (in any programmable way) on any desired user model information. In this paper we concentrate on GALE [21], the adaptation engine we renamed to the "Generic Adaptation Language and Engine". We describe the key elements that make GALE into a truly generic and highly extensible Web-based adaptive delivery environment.

Keywords: adaptation engine, adaptation rules, generic architecture.

1 Introduction

Vannevar Bush is often said to be the inventor of *hypertext* because of his article "As We May Think" [11]. (The term "hypertext" was not used by Bush and introduced much later by Ted Nelson [19].) Bush actually did more in that article, by envisioning that the user would build "*trails of his interest through the maze of materials available to him*". This type of user was called the "trailblazer". It is a first sign of personalization, aimed at facilitating *revisiting* information (either by the same user or by someone else). Another form of personalization was the addition of annotations: "*he inserts a page of longhand analysis of his own*". Adaptive hypermedia research, first summarized by Brusilovsky in 1996 [7] and updated in 2001 [8] aims at automating this trailblazing and annotating through *link adaptation* and *content adaptation*. Knutov et al [18] 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

[1] GRAPPLE stands for Generic Responsive Adaptive Personalized Learning Environment.

of dealing with *ontologies*, *open corpus adaptation*, *group adaptation*, *information retrieval* and *data mining*, *higher order adaptation*, *context awareness* and *multimedia adaptation.*

GALE [21] tackles some of Knutov's challenges directly through implemented functionality (that we will describe) and makes it possible to tackle more challenges by means of a highly extensible modular architecture. GALE builds on the experience we gained in the past when developing and using AHA! [3, 4, 6], the first general-purpose open source adaptive Web-server extension, used in different institutes all over the world.

This paper is organized as follows: Section 2 presents some background on adaptive Web-based systems, and positions GALE as a *generic* adaptive Web-server extension. Section 3 describes the GALE architecture, stressing the modular construction of communicating components. It also describes how GALE executes adaptation rules. Section 4 describes the adaptation language GAM, a simple textual representation of adaptation models. Section 5 concludes and points to needed and planned future work.

2 A Brief Overview of Adaptive Applications and Platforms

It is impossible to give a complete overview of the introduction of *adaptive functionality* in Web-based systems and applications. We will briefly mention some *influential* developments.

The fields of *intelligent tutoring systems* and *hypermedia* came together when the Web evolved to the point where it became feasible to add enough functionality to the Web server back-end. The award-winning ELM-ART[2] adaptive Lisp course [10] not only paved the way for later developments but also set a standard for *adaptive link annotations* by employing a "traffic light metaphor" of using green and red balls (and some intermediate colors like white and yellow) to indicate the "status" of links to course topics or pages. This metaphor was inherited by many later systems and can still be seen in some GALE applications as well.

When Brusilovsky created Interbook [9] he aimed at creating a platform that could be used for many courses (instead of the single Lisp course offered by ELM-ART). He used Microsoft Word as the authoring platform. An author would essentially write a textbook in Word. Fragments (paragraphs or sections) of text would be associated with some *outcome concepts* by means of a structured comment, and concepts would be indicated as being a *prerequisite* for other concepts, also by means of a comment. Converting a textbook (written in Word) into an adaptive on-line course was reduced to little more than a press of a button.

Adaptation is always based on information the system has about its user. Therefore *user modeling* is a key component in every adaptive system. Typically user modeling is based on rules that "translate" user actions into user

[2] ELM-ART stands for Episodic Learner Model, the Adaptive Remote Tutor.

information. Reading a course page means that you learn about a concept. That knowledge can be confirmed through a test. That knowledge is also used to check whether you satisfy *prerequisites* for studying other concepts. Knowledge levels of small concepts also "add up" to knowledge about chapters and whole courses. This description may give the false impression that an adaptive system really "knows" what goes on in the user's mind, with absolute certainty. There is however an alternative approach to user modeling, using Bayesian networks. This approach is taken by KBS-Hyperbook [16] for instance. User actions change the "belief" of the system that the user has certain knowledge. It is easier to deal with positive and negative "evidence" of the user's knowledge (or interest or any other type of information) in Bayesian networks than in systems that just use (event-condition-action) rules to update knowledge levels.

Recent versions of AHA! (the Adaptive Hypermedia Architecture) [3, 6] allow an author to define arbitrary *adaptation rules* (in fact *user model update rules*) and can thus support any model or interpretation of the information stored about users. Also, unlike Interbook's use of Microsoft Word for authoring, applications in AHA! are authored in HTML, and of course also delivered as HTML. This means that authors familiar with creating HTML can give an AHA! application any look and feel they desire. AHA! can for instance completely mimic the behavior and presentation of Interbook [5]. In order to be able to manage the complexity of having *arbitrary adaptation rules* and *arbitrary presentation* AHA! had to separate the authoring of the adaptation from the authoring of the content and presentation. Defining complex adaptation rules requires very different skills from writing the content of a course. We will see this in GALE as well.

ELM-ART, Interbook and AHA! are examples of systems that support *learning through adaptive information exploration*. The user receives guidance (through link annotation and in AHA! also through the *conditional inclusion of fragments*) but can browse through a course text in any desired way. More support for the *process* of navigation and the use of *services* (going beyond the single step of accessing a page) can be found for instance in APeLS, designed by Conlan and Wade et al [12].

Although adaptive applications in the field of elearning or "technology--enhanced learning" are the most common, there is also significant research in other application areas. Personalized advertising is becoming big business. This is highly visible in Google's personalized ads, but also in recommendations on shopping sites, like the "*people who bought this... also bought...*" messages on Amazon. Museums and other cultural institutes are offering adaptive personalized previews and guided tours or are studying this. The Dutch CATCH (Continuous Access to Cultural Heritage) research program has spawned a lot of research to improve access to cultural information. In one of its projects: CHIP[3] (Cultural Heritage Information Presentation and Personalization) a personalized art recommender was built for (and jointly with) the Rijksmuseum [22], as well as personalized virtual and physical (mobile) guided tours [13]. Italy is also strong in the adaptive access to information about culture and tourism, for

[3] See http://www.chip-project.org/ for more information and demos.

instance through UbiquiTO, a multi-device adaptive guide for Torino [2] and the work on Intelligent Inferfaces for Museum visitors from FBK in Trento [18]. The world of entertainment is following suit with personalized TV guides such as iFanzy[4] [1], a result of the ITEA Passepartout project.

The main difference between the mainstream adaptive educational applications and the others is the role of the *author*. Systems for culture, entertainment and business are typically large special-purpose Web-based Information Systems with an added personalization or adaptation component. There is also a body of research towards making Adaptive Web-based Information Systems more generic [16, 20]. The definition of the adaptation in Adaptive WIS is mostly automatically generated from semantic structures in the databases. In elearning a human author is involved in defining the adaptation at the "instance level", specifying in detail how which action of the learner leads to which change in the user model and how the user model state influences the adaptation. But this is not only the case in elearning. Adaptive (adventure) games for instance also require carefully crafted adaptation rules. GALE attempts to offer a *generic* solution to *authors* of adaptive websites. In the GRAPPLE project (EU FP7 STREP) the *adaptation language* and the graphical *authoring tools* played an important role. The GALE *engine* can be used to serve information where the adaptation rules are generated from (semantic) database structures, but in this paper we concentrate on manually authored applications.

3 The GALE Architecture

Figure 1 below shows the global architecture of GALE. A more elaborate description of GALE can be found in [21]. In this paper we can only present a brief summary of GALE.

Because of lack of space we will concentrate on the *processor stack* (bottom left of the figure) and the distributed way of *executing adaptation rules* (a collaboration between the adaptation engine, domain model and user model services. The former is responsible for performing the actual adaptation and the latter for performing user model updates.

3.1 GALE Processors and Modules

In GALE you can configure a set of *processors* that may operate on an outstanding request. Each processor is only active when the request has reached a state in which it can act upon it, and when finished it updates that state in order to notify the next processor that it can start processing the request. The actual adaptation functionality of GALE can thus be extended in arbitrary ways by adding processors to the stack. Here we briefly explain how a typical (http) request for a concept goes through the stack. We omit details like determining a layout for the presentation and logging for later use with data mining tools.

[4] See http://www.ifanzy.nl/.

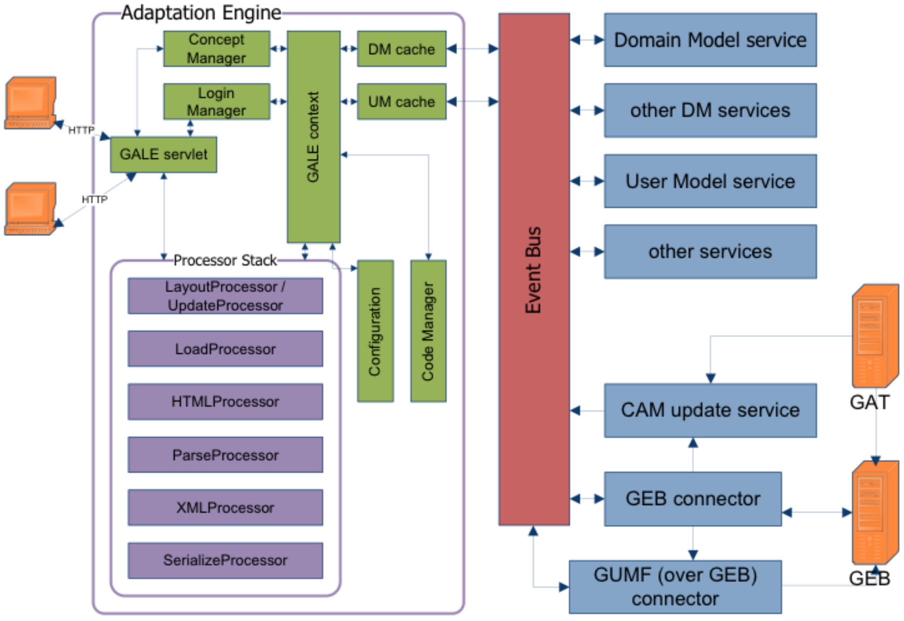

Fig. 1. The GALE architecture

1. When the top left part of the architecture has performed its duties of ensuring that the user is identified and a session is created the first processor to "touch" the request is the *UpdateProcessor*. It signals an *EventManager* that the "access concept" event has occurred. The default *EventAccessHandler* executes the *event code* of the requested concept, as defined in the application's "Domain Model" or DM. This part is concerned with updating the user model and will be described later. The *UpdateProcessor* will wait for the user model (UM) updates to be completed, so as to ensure that any further processing of the event is based on the *new* user model state. (For instance, when a learner requests a concept from a course, the knowledge update that should follow from reading the page associated with the concept is already performed before the page is retrieved, adapted and presented to the learner. It is important for an *adaptation rule designer* to know this order of events.
2. After the UM updates have been performed the *LoadProcessor* will retrieve the actual resource (file) associated with the concept. The name or url of resource may be dependent on UM, which is why UM updates must come first. The *LoadProcessor* retrieves the file (possibly by issuing an http request to a remote server) and creates an *InputStream* that can be used by subsequent processors to load and process the data. File name extensions are used to determine the mime type of the resource, and this may tell processors whether they should handle the request or not. At this point GALE could be extended with processors to handle all kinds of input formats, like images,

drawings, audio, video, etc. but by default GALE only provides processors to deal with HTML and XML.

3. Although GALE can handle HTML it really works with XHTML. When encountering HTML the *HTMLProcessor* uses the (open source) Tagsoup[5] converter to convert the file to XHTML. It creates a new *InputStream* that contains valid XHTML.
4. If the input is XML (also XHTML) the *ParseProcessor* converts the input into an in-memory DOM tree, using the open source dom4j[6] parser.
5. 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*. GALE can be configured to associate different modules with different XML tags. The default modules are targeted towards handling XHTML, but any module for handling any kind of adaptation to elements associated with any tag can be added, thus facilitating adaptation to very different types of XML documents, like MusicXML, SMIL, etc. Modules can change the tag name, attributes of tags and the content of the XML elements. (We give some examples later.)
6. The *SerializeProcessor* converts the DOM tree back into the textual XML format and presents that to the *GaleServlet* as an *InputStream* so that it can be sent to the user's browser (as an http response).

In GALE *Modules* are associated with XML tags in order to perform adaptation. Because of space limitations we only explain a few modules here:

– The *IfModule* handles the <if> tag. It expects <if> to have an argument "expr" that is a Boolean expression in GALE code (see Sect. 4.) 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 [17].
– 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. The link adaptation consists of (optionally): adding an icon in front of the link anchor (e.g. a colored ball to implement the "traffic light metaphor"), adding an icon after the link anchor, and adding a "class" attribute to the <a> tag, which in combination with a style sheet changes the presentation of the link. The default stylesheet uses three classes: GOOD, NEUTRAL and BAD and associates these with the colors blue, purple and black, just like AHA! did. Unlike in AHA! the number and choice of classes, colors, icons and conditions for using which class are all unlimited and easily configurable.
– The *VariableModule* replaces the <variable> element by a variable form the user model or the result of an expression. The *AttrVariableModule* does the same for an attribute in the parent tag. In XML the attributes of a tag cannot contain XML tags, so using a <variable> tag inside an XML tag to

[5] See ccil.org/~cowan/XML/tagsoup/ for more information.
[6] See dom4j.sourceforge.net for more information on dom4j.

make an attribute adaptive is not allowed. To make the name of an image in the <img> HTML tag adaptive for instance you can write:

<img><attr-variable name="src" expr="..."></img>

where the expression (not filled in here) results in the name for the image.

3.2 The Execution of GALE Adaptation Rules

As Fig. 1 shows GALE has an "internal" Event Bus through which different components communicate with each other. There are two essential sources of information for adaptive information delivery: the *Domain Model* (DM) that describes the conceptual structure of an adaptive application and the *User Model* (UM) that stores all the information the system can gather about its users. DM and UM services in GALE are separate services that can (if desired) run on different machines.

The DM of an application consists of *concepts* and *relationships*. Concepts have properties, including a title and description but also names (urls) of resources and conditions for selecting which resource. Concepts also have associated *event code* that is executed by the adaptation engine when a user accesses the concept. To minimize communication the adaptation engine maintains a cache of the DMs of the applications it serves.

The UM contains for each user some (arbitrary) personal information and for each concept of each application the user has accessed it contains *event code and* some attribute values. (The *event code* is stored only once but the attribute values are stored for each user.) Since the event code may make use of DM information the UM service has a cache of the DM. And since the adaptation engine needs to frequently access UM information it has a cache of the UM data of its (active) users.

Figure 2 shows how GALE handles UM updates caused by internal *event code* and UM updates received from external sources, such as the GUMF user model service that is part of the GRAPPLE framework.

When *event code* associated with a concept (access) causes a UM update (for instance the access to a concept means that the user gains knowledge about the concept) that update changes an attribute-value in the UMCache. The UMCache wishes to synchronize this update with the UMService and sends a "setum" message to UMService through the EventBus. The change to the attribute-value may trigger *event code* associated with the attribute. This event code is executed within the UMService and results in more UM updates. These updates must be synchronized with the UM cache in the adaptation engine so the set of changes os returned to the UMCache. When the UMCache sends out some updates it always waits for "results" to come back, because the adaptation process must work with the *new* UM state.

When an external UM service sends a UM update (as a "setum" message) this may also cause additional updates, and an "updateum" going to the UMCache. GALE can thus handle UM updates arriving at any time from any service.

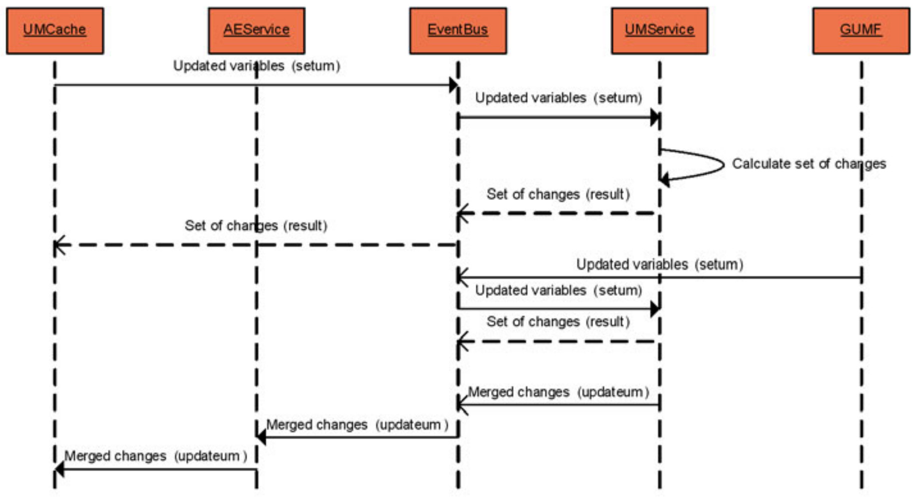
UMCache
AEService
EventBus
UMService
GUMF
Updated variables (setum)
Updated variables (setum)
Calculate set of changes
Set of changes (result)
Set of changes (result)
Updated variables (setum)
Updated variables (setum)
Set of changes (result)
Merged changes (updateum)
Merged changes (updateum)
Merged changes (updateum)

Fig. 2. The process of handling UM updates

4 GAM: The GALE Adaptation Model (GALE Code)

Creating an adaptive application requires that an author or adaptation designer defines *adaptation rules*. This process has been the Achilles heel of adaptive application design from the start. In Interbook [9] and early AHA! versions [4] an author would simply specify *prerequisite* and *outcome* relationships between concepts and the system would automatically generate adaptation rules that resulted in **the** adaptive behavior of the system. Authoring was easy, and flexibility was non-existent. As more flexibility was introduced, for instance in AHA! version 3 [6] a dilemma was born between simplifying authoring through graphical authoring tools [3] and empowering the author through a rich adaptation language. In the GRAPPLE project a graphical authoring environment was created [14] which allowed for easy adaptation design using templates (pedagogical relationship types) while empowering the author by making it possible to design arbitrarily many and complex new templates. In GRAPPLE it is thus possible to do little more than use *prerequisite* and *outcome* relationships, and at the same time it is possible to design the most complex adaptation you can imagine. Here we do not describe the graphical tool but only the underlying GAM language that can also easily be used by an author in a purely textual fashion. We introduce GAM by example (as a complete definition with examples would be far too long).

A GAM definition can be for a single concept or a number of concepts, and can be combined with (X)HTML content as well. Below is an example concept definition[7] for a concept http://gale.win.tue.nl/elearning.xhtml (that can of course be referred to as elearning.xhtml by other concepts on the same server):

[7] We omit some of the "escaping" of < and > symbols that is actually required by the XML syntax.

```
<?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} && ${#read} < 35)
#{#read, 35};‘}
#knowledge:Integer ¡ avg(new Object[]
{${<=(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>
```

The example code has the following semantics:

– #[visited]:Integer ‘0‘ means that this concept has a UM attribute called “visited”; it is an integer and is initialized with the value 0. The brackets [. . .] indicate that the value of this attribute is *stored* permanently.
– The code event ‘#{#visited, ${#visited}+1};‘ } means that when the concept is accessed the value of the “visited” attribute in increased by 1.
– When the value of “visited” changes its *event code* is executed which updates the “read” attribute.
– The attribute “read” is also an integer; it is also *stored* or *persistent*.
– The attribute “knowledge” is an integer which is not stored but calculated from the “read” value and the list of “knowledge” values of the children of the “elearning” concept.
– The attribute “suitability” is a Boolean, which is “true” by default. This too is not stored but calculated when needed. If there were prerequisites for the “elearning” concept there would be an expression that defines the condition for the concept to become suitable.

Another concept 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= {->(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 may 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> '
}
```

Again we do not explain this code but just illustrate that code can be shared between different concepts/pages, and can be placed in individual files or combined into a single GAM file. An adaptation designer can create the file “elearning.xhtml” and an application (or course) designer can use the adaptation defines by the designer by simply “extending” that definition and by defining relationships (like “parent”) between concepts.

When authoring through GRAPPLE's graphical tool similar templates are used, but instead of "extending" concepts to inherit attribute definitions and adaptation rules templates are "instantiated" (through copying). In terms of adaptive functionality this makes no difference.

The event code in GAM is essentially arbitrary Java code, in which some shorthand notation is used to refer to *properties* and *attributes* of concepts. The shorthand can be summarized as follows:

– Attributes are accessed by using # and properties by using? as part of the syntax of URIs. http://gale.win.tue.nl/someconcept#knowledge refers to the knowledge value for "someconcept" for the current user and #knowledge refers to the knowledge value for the concept to which the code is associated. http://gale.win.tue.nl/someconcept?title refers to the title property of the concept.
– someconcept->(somerelation) represents the list of concepts to which "someconcept"' has the relation "somerelation".
– someconcept<-(somerelation) represents the list of concepts that have a "somerelation" relation to "someconcept".
– ${#knowledge} is the syntax used to *retrieve* the value of the knowledge attribute.
– #{#knowledge, 100} is the syntax used to *set* the value of the knowledge attribute (to 100 in this example).

Using this explanation we can now understand the Java code

```
GaleUtil.avg(new Object[]
{${<-(parent)#knowledge},${#read}}).intValue(),
```

which executes a GALE utility method "avg" on the list composed of all the "knowledge" values of all concepts with a parent relationship to the current concept and the "read" value of the current concept, and which then returns this value rounded to the nearest integer. Such expressions can be used not only in concept and adaptation rule definitions but also in pages, for instance in the expression of an <if> tag. Typically these expressions would be simple, like <if expr="${#visited}==1"><then>This appears on the first visit only </then></if>.

The easiest way to understand and create GAM adaptation code is to look at examples and tutorial material from the GALE website gale.win.tue.nl.

5 Conclusions and Further Work

In the quest for the ultimate powerful, flexible and easy to use adaptive application (authoring and delivery) platform GALE is the most recent attempt. The research area of adaptive Web-based hypermedia systems has evolved from easy to use but very rigid systems (e.g. Interbook [9]) to powerful, flexible and extensible systems (first AHA! [3, 4, 6], now GALE) for which unleashing its power and offering very simple authoring environments is an immense challenge.

In this paper we have explained the modular and highly configurable and extensible architecture of GALE and the powerful GAM adaptation language, and have shown how we have attempted to keep authoring simple by reusing adaptation definitions created by others. Within the GRAPPLE project a graphical authoring environment was created as well. In the future (planned before the presentation of this paper) we will investigate how the graphical and textual authoring methods compare in terms of acceptance by authors. As part of this process more templates (for both authoring environments) will be created and more modules and processors added to GALE as well.

GALE is already being used by researchers in different parts of the world and by companies wishing to start delivering adaptive training material. This will inevitably lead to the discovery of new requirements for more powerful functionality for the next generation adaptive systems, so the quest for the ultimate adaptive system continures.

Acknowledgments. This research was supported by The EU FP7 project GRAPPLE, ICT-2007.4.1, project 215434. Earlier research on AHA! was supported by the NLnet Foundation.

References

1. Akkermans, P., Aroyo, L., Bellekens, P.: iFanzy: Personalised Filtering Using Semantically Enriched TV-Anytime Content. In: Demo at the Third European Semantic Web Conference (2006)
2. Amendola, I., Cena, F., Console, L., Crevola, A., Gena, C., Goy, A., Modeo, S., Perrero, M., Torre, I., Toso, A.: UbiquiTO: A Multi-device Adaptive Guide. In: Brewster, S., Dunlop, M.D. (eds.) Mobile HCI 2004. LNCS, vol. 3160, pp. 409–414. Springer, Heidelberg (2004)
3. De Bra, P., Aerts, A., Berden, B., De Lange, B., Rousseau, B., Santic, T., Smits, D., Stash, N.: AHA! The Adaptive Hypermedia Architecture. In: Proceedings of the Fourteenth ACM Hypertext Conference, pp. 81–84. ACM Press (2003)
4. De Bra, P., Calvi, L.: AHA! An open Adaptive Hypermedia Architecture. New Review of Hypermedia and Multimedia 4, 115–139 (1998)
5. De Bra, P., Santic, T., Brusilovsky, P.: AHA! meets Interbook and more... In: Proceedings of the AACE ELearn 2003 Conference, pp. 57–64 (2003)
6. De Bra, P., Smits, D., Stash, N.: The Design of AHA! In: Proceedings of the Seventeenth ACM Conference on Hypertext and Hypermedia, pp. 133–134. ACM Press (2006), `http://aha.win.tue.nl/ahadesign/`
7. Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. In: User Modeling and User-Adapted Interaction, vol. 6, pp. 87–129. Kluwer Academic Publishers (1996)
8. Brusilovsky, P.: Adaptive Hypermedia. In: User Modeling and User Adapted Interaction, vol. 11, pp. 87–110. Kluwer Academic Publishers (2001)
9. Brusilovsky, P., Eklund, J., Schwarz, E.: Web-based education for all: A tool for developing adaptive courseware. In: Computer Networks and ISDN Systems, Proceedings of the 7th Int. World Wide Web Conference, vol. 30(1-7), pp. 291–300 (1998)

10. Brusilovsky, P., Schwarz, E.W., Weber, G.: ELM-ART: An Intelligent Tutoring System on World Wide Web. In: Lesgold, A.M., Frasson, C., Gauthier, G. (eds.) ITS 1996. LNCS, vol. 1086, pp. 261–269. Springer, Heidelberg (1996)
11. Bush, V.: As We May Think. The Atlantic Monthly (1945)
12. Conlan, O., Wade, V.P.: Evaluation of APeLS – An Adaptive eLearning Service Based on the Multi-model, Metadata-Driven Approach. In: De Bra, P.M.E., Nejdl, W. (eds.) AH 2004. LNCS, vol. 3137, pp. 291–295. Springer, Heidelberg (2004)
13. van Hage, W.R., Stash, N., Wang, Y., Aroyo, L.M.: Finding Your Way Through the Rijksmuseum with an Adaptive Mobile Museum Guide. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010. LNCS, vol. 6088, pp. 46–59. Springer, Heidelberg (2010)
14. Hendrix, M., Cristea, A.I.: Design of the CAM model and authoring tool. In: A3H: 7th International Workshop on Authoring of Adaptive and Adaptable Hypermedia Workshop, 4th European Conference on Technology-Enhanced Learning (2009)
15. Henze, N.: Adaptive hyperbooks: Adaptation for project-based learning resources. PhD Dissertation, University of Hannover (2000)
16. Houben, G.J., van der Sluijs, K., Barna, P., Broekstra, J., Casteleyn, S., Fiala, Z., Frasincar, F.: Hera. In: Web Engineering: Modeling and Implementing Web Applications. Human-Computer Interaction Series, pp. 263–301. Springer, Heidelberg (2008)
17. Knutov, E., De Bra, P.M.E., 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)
18. Kuflik, T., Stock, O., Zancanaro, M., Gorfinke, A., Jbara, S., Kats, S., Sheidin, J., Kashtan, N.: A Visitor's Guide in an Active Museum: Presentations, Communications, and Reflection. ACM Journal on Computing and Cultural Heritage 3(3) (2011)
19. Nelson, T.: The Hypertext. In: Proc. World Documentation Federation Conf. (1965)
20. Rossi, G., Schwabe, D.: Modeling and Implementing Web Applications with Oohdm. In: Modeling and Implementing Web Applications. Human-Computer Interaction Series, pp. 109–155. Springer, Heidelberg (2008)
21. Smits, D., De Bra, P.: GALE: A Highly Extensible Adaptive Hypermedia Engine. In: Proceedings of the Twenty-Second ACM Hypertext Conference, pp. 63–72. ACM Press (2011)
22. Wang, Y., Stash, N., Aroyo, L., Gorgels, P., Rutledge, L., Schreiber, G.: Recommendations Based on Semantically-enriched Museum Collections. Journal of Web Semantics: Science, Services and Agents on the World Wide Web 6(4), 43–50 (2008)