

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

**Authoring through
Concept Structure Level Translation
of Adaptive Hypermedia systems**

By
E.W.A. Ramp

Supervisors:

Paul de Bra (TU/e)
Peter Brusilovsky (Pitt)

Eindhoven, June 2005

Abstract

In the last two decennia, many adaptive hypermedia systems (AHS) have been created. However, a number of these is not being further developed anymore, and is “wasting away”. Newer systems often have a more modern set of functionality, although sometimes they would benefit from obtaining some of the functionalities of the classic systems. In this thesis we will discuss the work we have done in the area of creating a reusable translation between a classic single-purpose AHS (InterBook) and a general-purpose AHS (AHA!), to address both mentioned issues: recovering/conserving the InterBook content by translating it to AHA!, and obtaining the InterBook content-authoring for AHA!, which has limited support in that area.

We have created the translation at a new level, the concept structure level, in contrast to previous attempts which created the translation at the code-level. We also had to translate/recreate the Look and Feel of the InterBook applications, for which we used the AHA! Layout Model.

Only obtaining the InterBook content-authoring for AHA! would cause a very limited support for the new AHA! functionality focusing on content adaptation. Therefore, we also extended the authoring process by adding support for some of these AHA! features. Thus we have added the functionality to AHA! to easily create new (large-content) applications.

Table of Contents

Abstract	3
Table of Contents	5
Introduction	7
Previous and Related work.....	8
1 (Classic) Single-Purpose AHS	9
1.1 InterBook.....	9
1.2 2L690	10
1.3 KBS Hyperbook	11
1.4 WHURLE.....	12
2 Multi-Purpose AHS.....	12
2.1 AHAM.....	13
2.1.1 Domain Model.....	13
2.1.2 User Model.....	14
2.1.3 Teaching Model.....	14
2.2 AHA!.....	14
2.3 KnowledgeTree	15
3 AHS to AHS Translation.....	17
3.1 Low Level Translations.....	17
3.2 Concept Structures	18
3.2.1 InterBook Concept Structure.....	18
3.2.2 AHA! Concept Structure.....	20
3.2.3 IatA Concept Structure.....	21
3.3 Look and Feel.....	24
3.3.1 InterBook.....	25
3.3.2 AHA!.....	27
3.3.3 IatA.....	28
3.4 Differences with InterBook.....	31
4 Authoring	32
4.1 AHA!.....	33
4.2 InterBook.....	33
4.2.1 Publishing Electronic Textbooks on the Web with InterBook.....	33
4.2.2 Advanced Authoring for InterBook	35
4.3 (Extended) IatA	36
4.3.1 New Concept Relations.....	37
4.3.2 Conditional Content	37
4.3.3 Reusable Contents	38
4.3.4 Internal Links to Contents and Concepts	39
5 Actual Translations and Encountered Problems	39
5.1 Graph and Concept Related problems.....	40
5.2 Other Problems.....	41
6 Remaining and Future Work	42
6.1 Optimizing Resemblance	42
6.2 Improving Usability	43
6.3 Improving InterBook/IatA.....	43
Conclusion.....	44
References.....	45
Appendix A. R2Net	47
Appendix B. Talking Bloom: AHA! Communicates with Sedona.....	48

Introduction

Can you imagine a world in which everything is exactly right for you? Would it not be nice if every clothing store offered a wide variety of clothing in just one size, namely yours? Would it not be nice if in every car the seats and mirrors would be fixed in the right position for you? Would it not be nice if all television channels offered only programs you like? The physical world is not likely to become like that because different people have very different needs, goals and desires. A lot of physical things need to be (manually) “adjusted” or “adapted”, like clothing, car seats and mirrors. But ultimately we want the world around us to *adapt itself* to us, to suit our specific needs. One “world” in which this is extra useful, and feasible to some extent, is the World Wide Web (WWW). Because of the huge amount of information available there, it is very hard to get exactly that part which you need, and, say, understand. But if the Web would “know” exactly who we are, what we know, what we want, and how we want it, it might actually adapt itself to us and provide us with the right information at the right time, in the right form.

Since the late eighties/early nineties researchers have been trying to address this issue by creating information systems that can indeed adapt themselves to the user. In the early years these were mainly separate attempts, but they turned gradually into more shared attempts, especially with Brusilovsky’s 1996 paper [9] that organized and listed all the work up to then.

Adaptive Hypermedia Systems (AHS) have evolved significantly since 1996. Many classics among those early AHS were developed to research some specific aspect, or had one specific purpose. Most systems focused on educational applications [3, 12, 16]. Although these classic single-purpose systems were definitely important to the evolution of the field, and many important results were obtained through these systems, they are now often wasting away, either because they are behind the current state of the art, or because their developers have moved on. Because of their single-purpose-ness it is hard to reuse them, or continue research with them.

InterBook [10, 12] is one of those classic single-purpose systems that have had their influence, and for which a lot of content has been developed. We will describe it in more detail at several places in this paper, starting with Section 1.1. However, because its researchers have moved on, InterBook has ended up on an old server, somewhere underneath someone’s desk, a fate similar to what probably many other systems of this kind have had bestowed upon them.

Nowadays, some of the focus has slightly shifted to more widely usable systems. More and more research is being done towards multi-purpose systems. Attempts like AHAM [4] have tried to set the stage for these systems, while AHA! [5, 7] and KnowledgeTree [14] have tried to create actual platforms to support this. However, one “loss” they cannot overcome on their own, is that of all the content (and effort to create it) for the classic systems. If we want to maintain this, we will need a way to use this content on new systems. For this, we will need to translate the original content in some way to make it run on the new system. Moving pure information content to a new system is often not hard to do. But in the case of adaptive systems the adaptation must be moved or translated to the new system as well.

This translation can be done in two different ways: the newer system can be changed to behave like the classic system by more or less merging the systems (on the program code level), or, if we have a new system that can support this, we can make the translation at a higher Concept Structure level.

AHA! supports these high-level translations with its separation of concepts and relations, and the exact implementation of these concepts and relations, all of which are defined separately from the core system. AHA! is described in more detail in Section 2.2.

A translation between AHS at the conceptual level is an important step towards separating the development of *adaptive hypermedia applications* from the development of *adaptive hypermedia systems*. Whereas our InterBook to AHA! translation shows that an InterBook application can be transformed into an AHA! application with essentially the same conceptual structure, a similar translation can also be used to provide an adaptation platform for applications developed with new authoring tools for which no corresponding delivery platform exists. An example of such an authoring environment is MOT [15].

The gain from these translations for the classic (source) system is quite clear, instead of having its efforts, knowledge and contents being wasted, it will find a new life on the new (target) system. The gain for the target system is less obvious. However, the target systems do not always have a superset of the features from the source systems; the translation might partially extend the target system too. Also, after the content is translated, the new system will often inherit the users of the source system, obtaining a wider/stronger user base.

This thesis project has a double intended goal. First, (through design and implementation) researching the feasibility of the translation of the classic InterBook system to AHA! using Concept-Structure translation in a reusable fashion, as to be able to translate all the current InterBook content automatically (we will discuss this in Section 3). Second, using this same reusable translation, we intended to obtain the easy authoring process from InterBook for AHA!, and extend this process to also support some AHA! specific features such as the reuse and conditional display of text fragments, for the (easy) creation of new AHA! applications (Section 4).

To facilitate the discussion of different AHS, which often have different terminology, we will use the terminology of our target system (AHA!) as much as possible, unless ambiguity requires a distinction between the discussed AHS and AHA! terminology, or when no adequate term is present in the AHA! system. Also, to distinguish between the original InterBook, basic AHA! and the emulation of InterBook at AHA! we introduce IatA (short for InterBook at AHA!) as name to refer to the emulation.

Previous and Related work

This project is not the first attempt to translate InterBook to AHA!. In [13] a low level translation is described, which (among others) gave rise to the extension of AHA! by a Layout Model. Concept relations were translated into low level event-condition-action rules, and a reverse translation is difficult, if not impossible, because the original relations were lost. However, since then, AHA! has evolved further, with the Layout Model now being an integral part of AHA! and a powerful adaptation rule language do define relations at a higher level. Our new translation will use this to optimally emulate the InterBook Layout and User Interface and functionality, with a minimal extension of the AHA! core code. That same project also created a first formula to emulate the InterBook Knowledge levels [6], which is reused in our project.

Other translations are also being worked on, including the mentioned MOT to AHA! [15] and a SCORM to AHA! translation. However, the former is a new authoring environment which uses AHA! as delivery platform, and the later is a (Sharable Content Object) Reference Model.

Although there is a definite resemblance in some areas, they are not significant, nor mature enough to influence our project.

1 (Classic) Single-Purpose AHS

Starting in the early nineties, adaptive hypermedia (AH) researchers have experimented increasingly with different functionalities and approaches. Often they would create a whole new system for each new research, resulting in many single-purpose systems. However, many of these classic systems have been surpassed by newer (single- or multi-) purpose systems. Because their number makes it impossible to discuss all classic and newer single purpose systems, we will limit ourselves to the discussion of only four single-purpose systems, roughly taken throughout the years. All four systems are meant for the presentation of (educational) adaptive electronic textbooks (although especially KBS Hyperbook also incorporates projects, but their purpose is the evaluation of the student knowledge, to perform better adaptation on the textbook). The systems we will discuss are the classic systems InterBook and 2L690, and the newer systems KBS Hyperbook and WHURLE.

1.1 InterBook

InterBook [10, 12] is a system dedicated to serving adaptive Electronic Textbooks (ET). As such, the user navigates through the contents of the ET (called interbook), and, by reading the pages of the ET, supposedly obtains (more) knowledge about the concepts, expressed in InterBook by multiple *knowledge levels*. One page might introduce multiple concepts, and a single concept may be introduced in several different pages. The (model of) student knowledge in InterBook is expressed in an overlay model of the concepts, rather than an overlay of pages. Reading one page might contribute knowledge to multiple concepts. Thus there is a clear separation between the contents and the concepts.

Content pages can have been “visited” or not, but their state influences the adaptation only in a very limited way. It is mainly the knowledge of concepts that is the basis for the entire adaptation of the system. Knowledge about the content pages is inferred through concepts, and in its turn, the knowledge is used to decide upon the exact adaptation that needs to be performed on the contents (up to whether or not the content is suitable for the user at all, represented by a colored bar above the content and colored bullets before the links), and on the amount of new knowledge “generated” when the user reads some content page. The concepts that are presented in a page, and for which new knowledge will be generated when a user visits that page, are designated as the *outcome* concepts of that page. The concepts that have to be known before the page is accessed are marked as prerequisite (or *background*) concepts. When a page is visited, InterBook increases the knowledge level of all its outcome concepts. The amount of newly generated knowledge is based on the existing knowledge about the prerequisite concepts, the way the outcome concept is presented in the content, the “visited” status of the page, and the starting knowledge of the outcome concepts. If the user lacks this prerequisite knowledge, much less new knowledge is generated when a user visits the content page, than when a user would visit that page while he already has all the prerequisite knowledge. On the other hand, when a user has already visited a page, subsequent visits will generate less knowledge increase.

The *concepts* are explicitly present in an interbook through the glossary. For each concept the glossary lists related pages (pages where the concept is either a prerequisite or an outcome), and a description (if available) which also increases knowledge when read. Also next to every *content page* is a listing of links to the related concepts (in the glossary).

The previously mentioned colored bar and bullets indicate the suitability of the content page or destination of the link, implementing *link annotation*. Red indicates not yet suitable, green suitable and white indicates neutral. Neutral is used for pages that offer *no new knowledge* in any way. Links to concepts can also be followed by one of three sizes of checkmarks, indicating the current knowledge level for that concept.

Direct guidance is available through a “*teach me*” button. It will add the current page to a stack, together with pages that need to be read to obtain all prerequisite knowledge. We will discuss the InterBook concept structure in Section 3.2.1, its user interface in Section 3.3.1 and its authoring process in Section 4.2.

1.2 2L690

Starting from a slightly different viewpoint than InterBook by “ignoring” structured textbooks, 2L690 (previously known as 2L670) [2, 3] focuses on hypertext documents, to which it adds adaptivity. It realizes both *adaptive content* and *adaptive (and annotated) linking*, through the use of conditionality and cascading style sheets (CSS). CSS are used to color links marked by “class=conditional” in a specific color, by default, “desired” (suitable) links are colored blue, “undesired” black, and visited (or neutral) purple. Links marked with “class=external” are represented in red. This color scheme can be overridden by both the author and the user. The choice for these default colors is not random; blue and purple links are used in normal hypertext systems (browsers) to indicate unvisited and visited links, the choice for black for undesired links is because this will make the link to “disappear” in the surrounding text, effectively implementing *link hiding*.

Another form of adaptive link hiding, *link removal*, is realized through the same means as adaptive content (or conditional content). First with C preprocessor commands, later with html comments (to increase readability of the html code) it is possible to create conditional contents. [3] gives two examples, one for conditional content in general, and the following for link removal through html comments:

```
<!-- if desired -->
  <a href="...">
<!-- endif -->
  here is the link anchor text
<!-- if desired -->
  </a>
<!-- endif -->
```

The only relation that is supported for pages (and fragments) is that of prerequisite. Certain knowledge is required (desired) before other pages become suitable. The domain model is an overlay of the content pages. A visit to a previously unvisited, suitable page generates knowledge for the accompanying concept.

All choices concerning the design of 2L690 were made with the notion in mind of using any html-editor to create (and annotate) the content, while still being able to use all HTML features including frames, scripting and java applets.

Many features of 2L690 are still visible in its posterity AHA!; the default color scheme, and conditional fragments are (be it through slightly different (authoring) techniques) still available in AHA! (see Section 2.2).

1.3 KBS Hyperbook

The database-inspired KBS Hyperbook [16, 18 ,21] has a similar goal as InterBook, presenting an automatically interlinked adaptive ET to the user (although a project based approach is followed), and also offers the same kinds of information, but presents them in a different layout. Similar to InterBook, KBS Hyperbook has *contents* and *related concepts*. The related concepts are listed in a similar fashion as the InterBook glossary, but are sorted by relation type. KBS Hyperbook also uses the colored bullets to indicate suitability (expressed as “already known”, “suggested” or “too difficult”) Figure 1 shows the User Interface for the KBS Hyper book system.

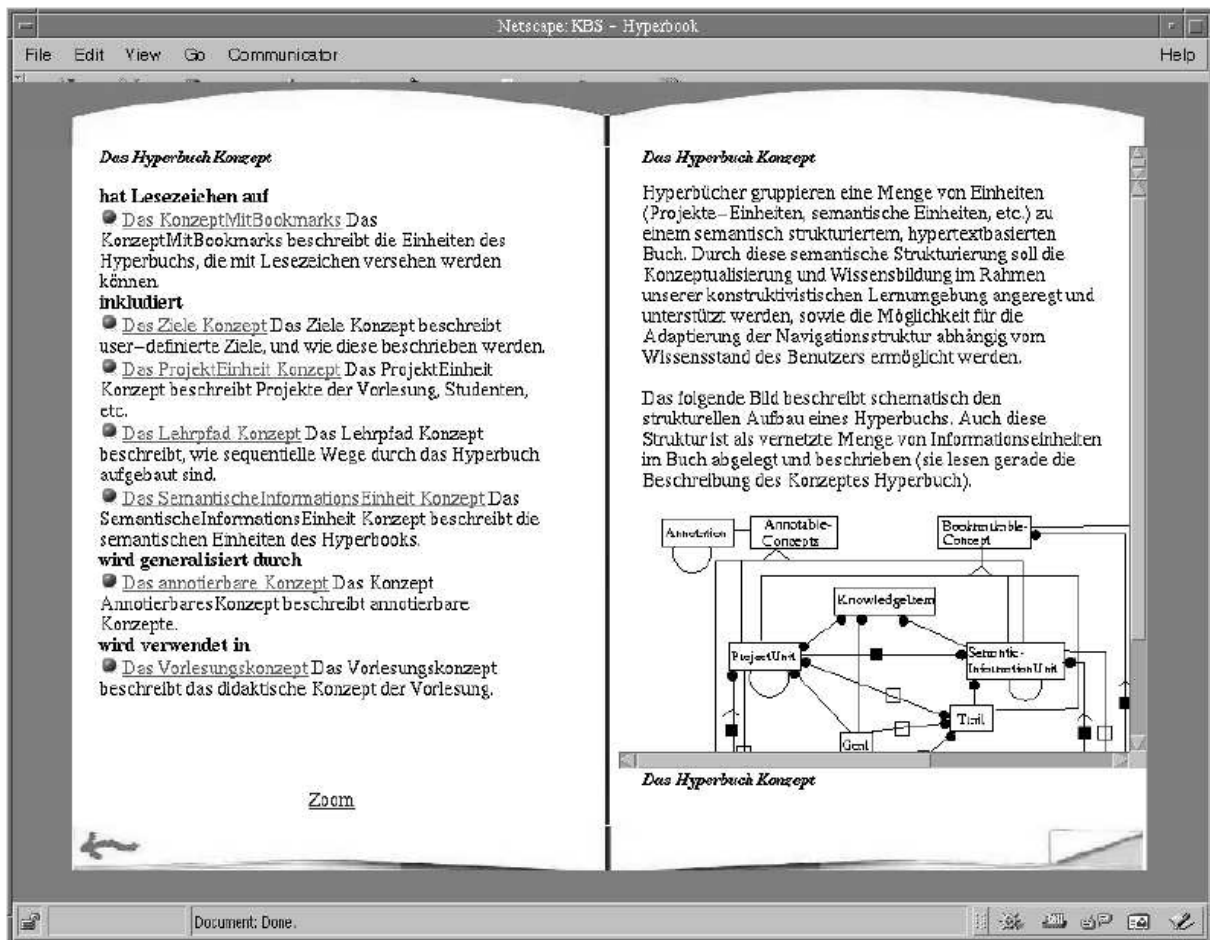


Figure 1. From [21]: a Hyperbook page, generated by the KBS Hyperbook system.

Just like the difference in layout, the approach to the contents is also different. KBS Hyperbook separates data into *information units*, which “do not correspond to syntactical parts of a book (such as sections or chapters)” [18]. These information units are then mapped onto concepts within a domain model. For each concept there is one main information unit, and possibly several other units in which it occurs, but not as main unit.

The domain model is built before the contents is created, and defines relations (of multiple types) between concepts. The content is “hooked into” the domain model afterwards in an open fashion; even during runtime it is possible, for e.g. students, to add additional pages to a Hyperbook.

The knowledge about concepts (called Knowledge Items, KI) and possible prerequisites are expressed in the user model (student model) through probability vectors; for every KI the probability is stored for the users knowledge about the KI, with four grade marks attached to certain values of *expert*, *advanced*, *beginner's*, or *newcomer's* knowledge. A Bayesian network is used to infer this probability. The information about the user is, unlike systems like InterBook or 2L690, not obtained through observing which pages the user has read, but by either (human) expert analysis of the user, or by questionnaires answered by the user at the end of projects done by the user.

Besides the annotated links, KBS Hyperbook has the ability to *generate trails to guide the user*, and to *automatically select suitable projects* for the user to do. It also can automatically select and recommend a (sub-)goal, based upon the user's knowledge (or missing knowledge). For this KBS Hyperbook will use a final goal of full knowledge of the whole Hyperbook.

1.4 WHURLE

A very recent AHS is WHURLE [8, 19]. It also focuses on adaptive learning content, similarly to the previously mentioned systems. A major difference is however that it focuses on content adaptation, as opposed to link adaptation. From the previously mentioned systems is only 2L690 capable of something similar. Where in 2L690 it is possible to filter out some content (through conditional fragments), in WHURLE the content is dynamically created by combining multiple separate fragments (*transclusion*).

The fragments (called *chunks*) are intended to be created by *subject experts*. These conceptual discrete units of contents can be anything from text fragments to images with captions and even assignments, and are combined into *lessons* by *teachers*. Creating lessons involves grouping relevant chunks into constrained constructs of hierarchical pages, together with a default pathway. *Technical authors* create the user (domain) model on which adaptation is performed.

The UM uses an *overlay model* and a *stereotype model*. The stereotype model is used to classify users, according to prior experience and ability, in novice, intermediate or advanced category. The overlay model is used to measure the user's knowledge for the current domain. Both models are used in adaptation filters, which define what contents is displayed. Both the teachers and the users can create (bi-directional) links, which are rendered into the final content pages.

2 Multi-Purpose AHS

In the previous section we discussed AHS that had a single-purpose. All mentioned systems have the purpose of presenting Adaptive Electronic Textbooks (ET) in an educational context. Nowadays more and more attention is being put to multi-purpose AHS. The exact approaches vary greatly, from the design of general reference models [4] to systems that can serve a great variety of different kinds of content, by moving away either from the traditional (adaptive) ET [14] or from the educational focus [5]. We will discuss examples of these three types in the following sections. Firstly, we will discuss AHAM (Section 2.1) which is a general reference model. Section 2.2 will introduce AHA!, the successor of 2L690 with a focus on universal reusability. An architecture that provides support for different types of adaptive Hypermedia is KnowledgeTree, which we will discuss in Section 2.3.

2.1 AHAM

In 1988 and 1990 a number of researchers and developers created the “Dexter model”. This model was designed to be a common reference model for Hypermedia systems. AHS have some features that do not fit in this Dexter model. For instance, AHS monitor the user and maintain a (permanent and continuously updated) user model; a record of the knowledge and interests from the user. The definition for the Dexter model [17] states that “by default, pending changes to instantiations are not saved” and that “the history is a sequence of all operations carried out since the *last open session operation*”. This and the placing of the history in the run-time layer implicitly disallow maintaining an inter-session history of user knowledge and actions, which is typical to AHS.

In 1999 several researchers tried to extend the Dexter model to create a reference model for AHS. The result was the Adaptive Hypermedia Application Model (AHAM [4]). Similar to the Dexter model, it focuses on the *presentation specifications*, *anchoring* and the *storage layer*. However, it deepens Dexter’s storage layer by distinguishing a *Domain model* a *Teaching model* and a *User Model* within this layer (see Figure 2), and defines the adaptation based on Brusilovsky’s observations in [9].

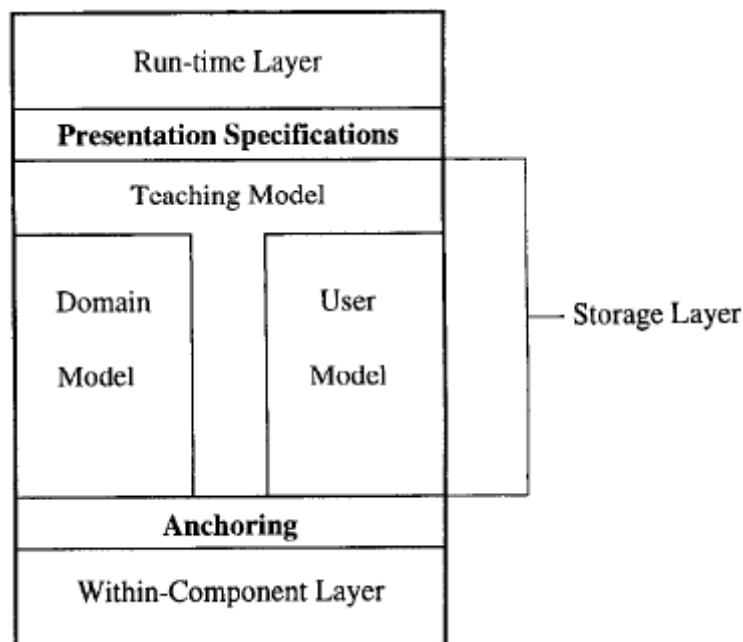


Figure 2. From [4]: the AHAM model

2.1.1 Domain Model

AHAM places most of Dexter’s storage layer functionality in what it calls the domain model. In short it represents the authors view on the application domain. As such, it defines *concepts* (Dexter: components) with unique identifiers (uid) and component information consisting of attribute value pairs, a sequence of anchors and a presentation specification.

In AHAM, there are two kinds of concepts, *atomic concepts*, which correspond to a single fragment of information from the within-component layer and *composite concept components*. These composite concepts have one attribute which is a *children attribute*; a sequence of either atomic, or other composite concepts. The former is called a page concept, the latter an abstract concept.

The sequence of anchors (or *relations*) connected to a concept is not limited to the Dexter notion of *link (-component)*, but can also be a *relation*, e.g. a prerequisite relation. The *functionality* of these relations is defined in the *teaching model*. AHAM allows in accordance with the Dexter model anchors with an arbitrary number of specifiers, and does not limit nor define specific relation types, but allows authors to “invent” new relation types as desired.

2.1.2 User Model

AHS maintain a permanent and continuously updated inter-session history of user actions, which represents how the user relates to the domain model. Because information in Dexter’s run-time layer does not carry over between sessions, and Dexter’s within-component layer concerns only “with the content and structure *within* the components”, the storage layer is the only correct place to define this *user model*.

In its universality, it stores information about the user. This information can be pages read, (inferred) knowledge, access times, etc., and as such will mostly be related to concepts (through their *uid*). AHAM does not define, nor limit either the exact information, or the value types.

2.1.3 Teaching Model

The combination of the *domain model* and the *user model* to determine the presentation of information is defined in the Teaching model. Its role seems twofold; it defines the adaptation performed on the content when it is presented to the user (e.g. link annotation), and it defines *suitability* (called *ready-to-read* in [4]) of concepts and accessory updates of the user model when the user visits the concept. However, these two functions are closely related, the suitability is often a base for the adaptation.

Suitability and user model updates are expressed in *concept relations* (called pedagogic rules in [4]). In AHAM these concept relations can be executed in one of two phases, either before or after the generation of the presentation of the adapted content. The following fragment from [4] is an example of the representation of the *functionality* of a *prerequisite relationship* CR, which expresses that for all prerequisite concepts (CR.ss[1]) the user needs to have a certain knowledge (CR.cinfo.required-knowledge[1]) before the concept (CR.ss[2]) becomes suitable:

```
<
CR.cinfo.type = prerequisite and CR.cinfo.dir[1] = FROM
and CR.cinfo.dir[2] = TO and CR.ss.length = 2 and
CR.ss[1].uid.knowledge-value ≥ CR.cinfo.required-
knowledge[1] ⇒ CR.ss.uid.ready-to-read := true, pre, true
>
```

For a full explanation we refer to [4].

2.2 AHA!

Being the successor of 2L690, AHA! [5, 7] has inherited several of its features. Its creators have moved on too however, into generalizing the system. The extension to allow authors to define their own concept relations (on top of low level event-condition-action rules) has at times given AHA! the designation as an *assembly language for adaptation* [5]. Also different from 2L690 is the Layout Model, with which any feasible layout is achievable. Because of this extensibility and configurability it is possible to create almost any kind of application.

Also because of this extensibility and configurability we will use the AHA! tutorial as *standard* or *basic* AHA! application; when we refer to basic AHA! we mean its functionality as it is being used by the tutorial.

New in AHA! is support for (reusable) conditional included fragments. These fragments depend on a condition (similar to conditional fragments) whether they should be included in a page, but, because they are defined separate from the page content, they are not limited to a single page, but can be included in multiple pages. When such a conditional included fragment is indeed included and displayed to the user, it will generate (some) knowledge for the according fragment concept. This is about 1/3rd of the knowledge generated by a page.

AHA! comes with the Graph Author; a supporting authoring tool to create a concept graph by graphically connecting (page- and fragment-)concepts through prerequisite (or other) relations. Figure 3 shows the Graph Author displaying the concept graph for the AHA! tutorial.

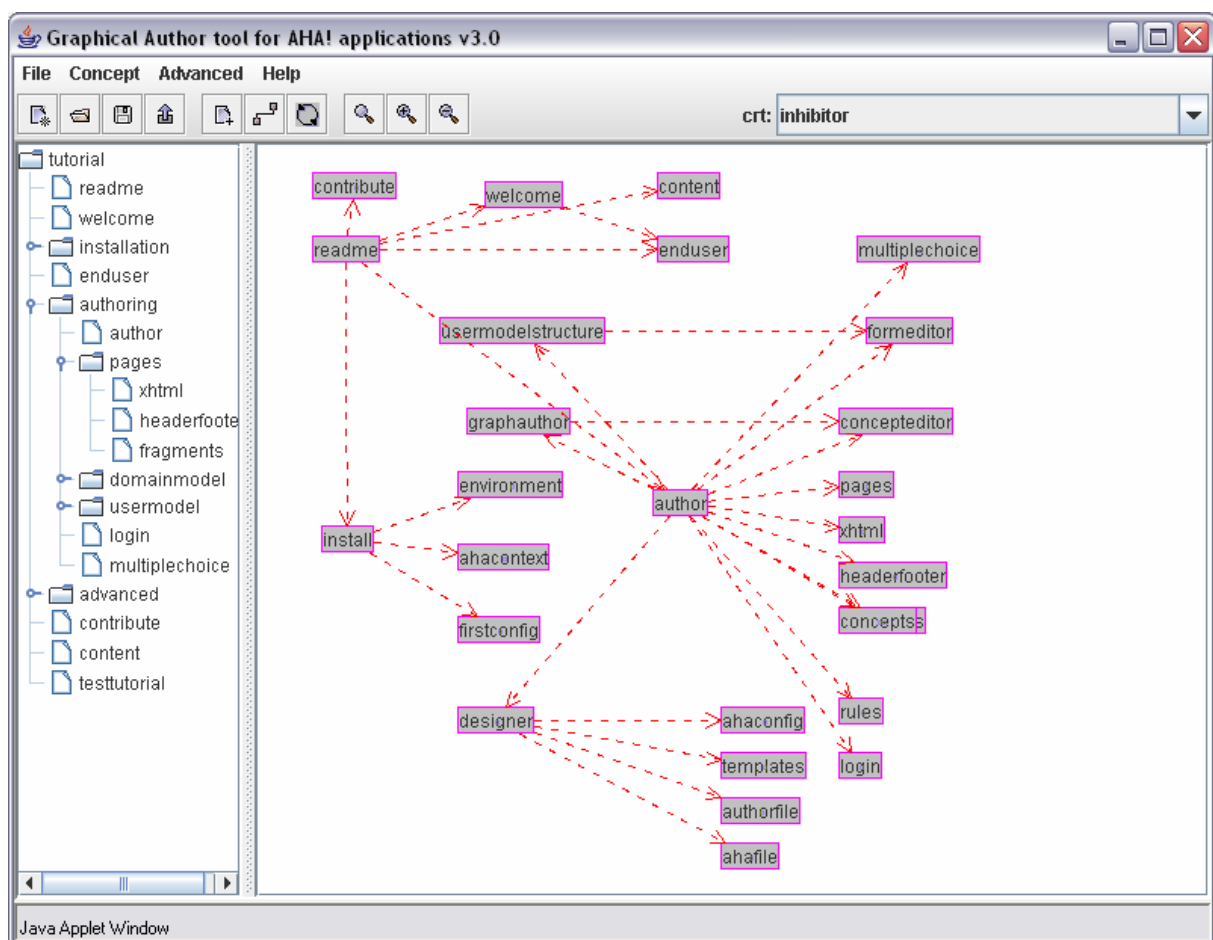


Figure 3. AHA! Graph Author Displaying the Graph for the AHA! Tutorial

We will introduce AHA!'s basic concept structure in Section 3.2.2, its user interface in Section 3.3.2 and discuss its authoring support in Section 4.1.

2.3 KnowledgeTree

Although KnowledgeTree focuses on E-Learning, it can still be considered a multi-purpose system. It is an architecture that integrates several reusable single-purpose (sub-)systems, and tries to create an adaptive alternative to learning management systems (LMS) as Blackboard,

while allowing for as much authoring freedom as possible for the teacher to design his own approach to teaching his class.

KnowledgeTree is a distributed architecture for adaptive E-learning based on the re-use of intelligent educational activities [14]. Its main components are *learning portals*, *activity servers*, *student modeling servers* and *value adding services*. One or more of each of these servers can be present in the architecture, and all have their own function, although servers that integrate several of these functions into one are also possible. Figure 4 gives a graphical representation of the architecture.

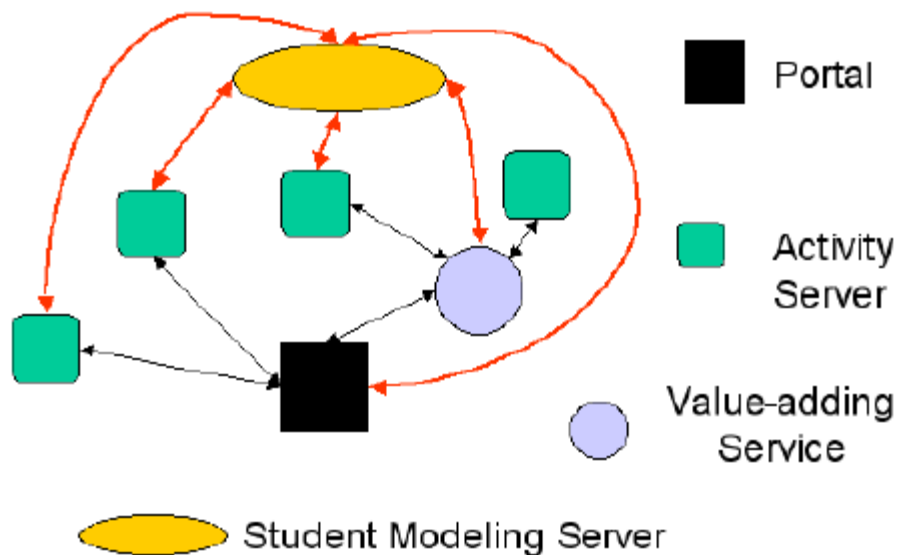


Figure 4. Main components of the KnowledgeTree distributed architecture (from [14])

The *learning portal* is the starting point for a student user. It provides a *centralized single-login point*. The teacher can structure the course content here with *reusable learning content* and *learning support services* (together called *activities*, located on an arbitrary number of activity servers), which is then served to the students, in a similar fashion as an LMS. The learning portal can query activity servers for relevant activities and launch these activities upon selection by the student or the portal self.

Activities accessible through the learning portal reside on (possibly multiple) content-specific *activity servers*. These serve the role of (reusable) content and service providers. The activities can be highly interactive, intelligent and adaptive, and can be referenced separately. Information about the student is obtained from and stored to the student modelling server.

A combination of functions of both the learning portal and the activity servers are available through *value-adding services*. Similar to the learning portal, they can query and access activities, and similar to the activity servers, they can be queried and accessed by learning portals. Using this structure, it is possible to create larger building blocks from multiple activities, which can be reused in different ways for different courses. The content from activity servers could be compared to *atomic* or *page concepts* in AHAM [4], while that from the value adding services is comparable with *abstract composite concepts*.

The *student model server* implements what could be seen as a combination of the *teaching model* and the *user model* from AHAM. It provides a centralized storage place for the user models, and is able to deduce knowledge from the stored information. It can be referenced from the other servers to obtain user information on which to perform adaptations.

The researchers have implemented two different portals and several activity servers and value-adding services [14], of a variety of functionality, however, because these separate servers and services can be considered single-purpose, we will not discuss them in further detail.

3 AHS to AHS Translation

When we want to translate an application from a special-purpose AHS to a generic AHS, the main goal is make the translated application work exactly like the original (or at least mimic the original as closely as possible). This means the target application performs exactly the same adaptation and user model updates as the original (under the same circumstances). To achieve this, there are two approaches.

The usual approach is to make a low level translation; the functionality of the source AHS is coded into the target AHS where it is missing the required functionality. We have used a slightly different approach for our first goal; using the extensibility of AHA! concept relations, we have created a translation from InterBook, based on the concept structure. For completeness we will give a short introduction to low level translations, followed by concept structure translation in general, and finally the details of our translation from the InterBook concept structure to AHA! in specific, by introducing the concept structures of both, and the resulting concept structure for IatA.

An AHS is more than only its concept structure, especially to its users. They interact (and have become familiar) with a certain *user interface*. Omitting to translate this look and feel would result in an unsuccessful translation from the user point of view. Therefore we will discuss this translation in a similar fashion as the concept structure; first we will introduce both the source and target system, followed by the resulting interface. We will conclude this Section with an evaluation of the differences with the source system, with respect to this translation.

3.1 Low Level Translations

In this project we do not consider the approach of extending the target system with special code to emulate the source system's specific adaptive behavior. We only consider the use of a powerful *adaptation rule language* of a generic AHS (AHA!) through which very different types of adaptive behavior can be created, and an *extensible layout model* to mimic the appearance of the original ET.

In [13] a first translation from InterBook to AHA! is described. This translation generated adaptation rules directly. For instance, if (as explained in Section 2.2.2) a page P has an outcome concept C, the translation would associate an adaptation rule (an event-condition-action rule) with the *access attribute* of P. The condition of that rule would be the sufficient *knowledge* of the background concepts for P, and the action would be an increment of the *knowledge attribute* of C. If the background concepts are known by the user the knowledge increase would be higher than when the background concepts are not (all) known. The translation would create and combine the adaptation rules for all the background and outcome concepts from the InterBook application. Because the conditions for background concepts and the knowledge update actions (conditionally) generated by outcome concepts can be represented exactly by AHA! adaptation rules the adaptive behavior of InterBook can be emulated completely by a translation to AHA!. However, the resulting AHA! application is difficult to understand by looking at the adaptation rules, because from these rules it is not

possible to reconstruct the original InterBook structures of background and outcome concepts. In the next Section we discuss a better approach that makes use of AHA!'s high-level structures (and its compiler to generate the low level adaptation rules that no longer need to be understood in order to understand an application's behavior).

3.2 Concept Structures

When applications (like InterBook electronic textbooks) are nicely structured, we wish to preserve that structure in the translation to another system. When applications can be described using a concept and concept relationship structure on the target system, we need only translate the source concept structure to the target system, and ensure that the *meaning* of the concepts and relationships is the same after the translation. In AHA! arbitrary *concept relationship types* can be defined, and bound to a generic translation into *adaptation rules*. The latter translation is completely local to AHA! and needs to be defined only once for the source system (InterBook in our case). It can then be used to translate all applications that exist on that source system (InterBook). The creation of the translation from source concept structure to target concept structure requires no knowledge of the internals of the target system or its adaptation rules. And the translation of the concept structure to adaptation rules on the target system requires no knowledge of the internals of the source system, only of the *meaning* of the concepts and relationships on the source system. In any case, no technical knowledge of the internals of either the source or the target system is needed to design and implement the translation process. The translation does require a target system that can process concepts structures in all their diversity, but this can be done as we will demonstrate with the translation of InterBook onto AHA!.

Unfortunately translating the conceptual structure is not enough. Different AHS associate different meaning and different behavior to the same conceptual structures. As an example, when a user reads a page for which some required background knowledge is missing, the standard InterBook behavior is to increase the user's knowledge value of the outcome concepts a bit every time the page is accessed. The standard AHA! behavior is to increase this knowledge a bit the first time the page is accessed but not a second time (until the page is visited again when the background knowledge is sufficient). It is clear that the adaptation rules for the knowledge updates when pages are accessed must be different to implement the InterBook or the AHA! behavior and probably different again to emulate some other AHS. To realize this, we use the AHA! *adaptation rule language* to define an alternative knowledge update for the InterBook outcome relationship.

We will first introduce the InterBook concept structure, followed by the basic AHA! concept structure for reference purposes. We will conclude with the (new) IatA concept structure, as we have realized it within AHA! and give a description of the new outcome relationship.

3.2.1 InterBook Concept Structure

As mentioned in Section 1.1, InterBook has a clear distinction between *concepts* and *contents* (or *pages*). Concepts are single entities, with a *knowledge* attribute. They define the domain of the ET. Knowledge for these concepts can be increased by reading about the concepts in the contents, by reading a description of the concept in the glossary, or through external application.

Contents, the original sections of a textbook, displayed as one textbook section at a time to the user, has one explicit attribute *visited*, indicating if the user has viewed the page already. It also has an implicit attribute, *suitability*, indicating whether the user has (at least the first level

of) knowledge of all prerequisite concepts or not. When the user views a page, knowledge is contributed to all concepts which have an outcome relationship from that page. InterBook uses three different names for the outcome relationship (outcome, exemplify and introduces), for organizational reasons; it allows to sort, index and annotate the links in the Glossary in a more specific way. However, this functionality was never further developed, and all three relationships currently have the same functionality:

- The *first* visit to a *suitable* page contributes *one full knowledge* level to all outcome concepts, although they can not increase beyond the second knowledge level.
- The 2nd to 4th visits to a suitable page contribute another *full knowledge level* (in total) to all outcome concepts, again with the maximum of the second knowledge level.
- Visiting a *non-suitable* page (if not all prerequisite concepts are known) increases knowledge only by 1/3rd of the increased amount compared to a visit to the same page when suitable.

There is also a third level of knowledge within InterBook, reserved for external input, e.g. from successful completion of tests. However, there are no concept relationships or other internal mechanism available for this, and therefore we will leave this outside our discussion of the InterBook concept structure.

Given this description of the InterBook approach, we can now express it in terms of concepts, attributes, and relationships similar to AHA!’s generic model. The first things we notice are the content and concepts, which perform different roles, although they have relations connecting them. They also have different attributes: content has a “visited” attribute, while concepts have a “knowledge” attribute. These attributes will be manipulated in the concept relationships as described.

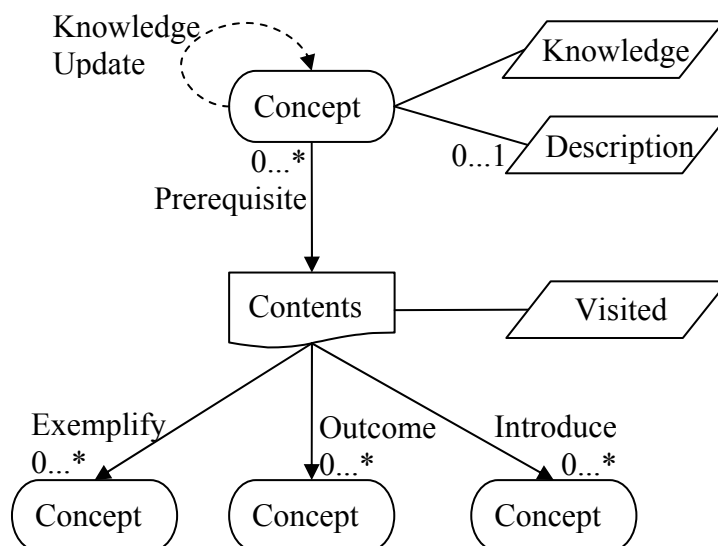


Figure 5. InterBook Concept Structure

The relationships are quite straightforward, we will use a *prerequisite relationship* to express that a page requires knowledge from certain concepts to be present before the (content of the) page becomes suitable. We will use separate *outcome*, *introduce*, and *exemplify relationships* to connect a page with concepts that are presented on the page in the three different ways. Any number of concepts can be connected to a single page through these relationships, as shown in Figure 5 by the “0..*”. The dashed Knowledge Update relationship indicates the knowledge increase (with a maximum of the first level) when a user visits the concept in the

glossary *and* there is a description associated with the concept. The knowledge attributes are omitted in the figure from the lower three concepts for simplicity.

3.2.2 AHA! Concept Structure

Unlike the InterBook concept structure, the (standard, or default) AHA! concept structure is only of limited interest to us. We will shortly discuss it here to be able to indicate the differences that we have to overcome, and similarities that we can use, when we try to translate the InterBook concept structure onto AHA!.

AHA! makes a different distinction between concepts and contents than InterBook: *contents* is an (optional) *attribute* to concepts. A user does not visit the contents, but visits the concept and subsequently gets to view the content. Because of this the *visited* attribute is connected to every concept, and every concept has a *knowledge* attribute. Concepts can be related to each other by a *prerequisite relationship*, besides that they are organized in a *hierarchy*. (See also the left side of Figure 3, earlier in this document). This hierarchical structure can be based similarly as in InterBook on *chapters and sections*, but can also express a *concept hierarchy* (comparable to part-of or is-a). This “implicit” relation between concepts is used within AHA! for *knowledge propagation*. If knowledge is obtained for one concept, it propagates (a part of) this knowledge to its parent concept, which, in its turn, might propagate part of that knowledge to its parent, etc. Each concept (except the root concept) has exactly one hierarchical parent, and therefore also exactly one “implicit” knowledge propagation relation.

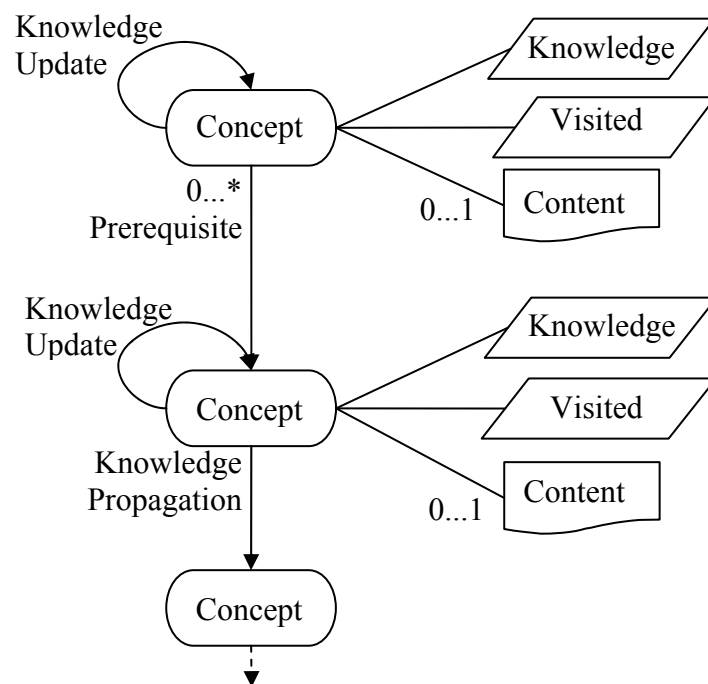


Figure 6. Basic AHA! Concept Structure

If a user visits a *suitable* concept, the knowledge for that concept is updated to “*known*” (internally expressed as integer 100). A visit to a *non-suitable* concept updates the concept’s knowledge to “*partially known*” (integer value 35). Part of the gained knowledge is then propagated to its parent. Figure 6 shows the above described basic AHA! concept structure with the attributes and relationships. Attributes for the bottom concept (supposedly the hierarchical parent of the middle concept) are omitted, as is the knowledge propagation

relation for the topmost concept. The dashed arrow indicates the possible further propagation of knowledge.

3.2.3 IatA Concept Structure

The InterBook concept structure and the AHA concept structure seem similar on first sight (Figure 5 and Figure 6), besides a difference in attributes. This is less important because the InterBook attributes are a subset of AHA!’s attributes, especially if we interpret the description attribute for InterBook concepts as content. However, they have actually one large difference. The functionality of their binary outcome relations is very different. Therefore, if we extend AHA! with a new outcome relation, the translation between the two concept structures should be possible. We will describe the new relation in Section 3.2.3.1

To realize the InterBook concept structure in AHA!, the easiest way is to identify similar structures, and build upon that. The easiest of these is the *InterBook contents*. By simply reusing AHA! concepts we obtain a *page concept* with a superset of the required attributes. Having contents being an attribute too is no problem, AHA! will take care of displaying this in a correct fashion. Furthermore, we can ignore the values of irrelevant attributes like the Knowledge attribute; which also indicates the importance of the “built-in” unary Knowledge Update relation; it is irrelevant, and its presence neither adds nor removes anything. Therefore, we will translate the InterBook contents in a 1 to 1 fashion to AHA! (page-)concepts.

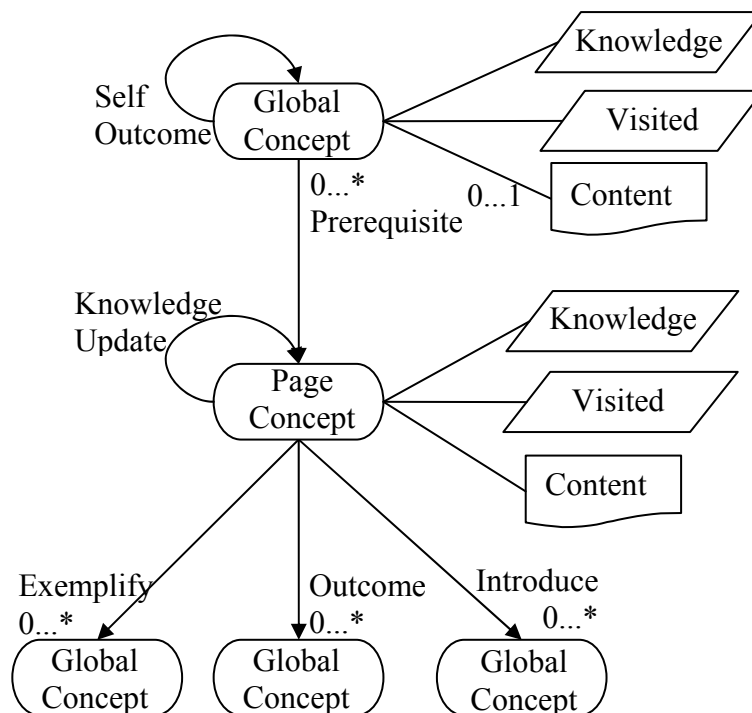


Figure 7. Translated InterBook Concept Structure

The (optional) description connected to *InterBook concepts* is nothing more than a short piece of contents, which is given a slightly different treatment within the InterBook system. Realizing this as (optional) contents within AHA! is the most straightforward approach. We will rely on AHA! to treat it in a different fashion (this is explained in more detail in the Section 3.3.3 on Look and Feel of IatA). The conditional InterBook Knowledge Update for concepts is too different from the AHA! Knowledge Update, even if we would ignore its conditional nature. Therefore we have created a new concept (designated *Global Concept*

because of its more global role within InterBook compared with page concepts), with a built-in unary relation that is based upon the newly created *outcome* relation. It is, however, limited to increasing knowledge to a maximum of the first level.

The prerequisite relation is nothing different for both AHS in its functionality. The only difference is the required amount of knowledge that should be known. This value is parameterized within AHA!, allowing us to entirely reuse this relation. The thus obtained concept structure (including the new relation discussed in the next section) is displayed in Figure 7.

3.2.3.1 Outcome Relationships in AHA!

The difference in functionality and use of the binary outcome relation in both AHS, required extending AHA! with a completely new relation. In InterBook knowledge is represented in 3 knowledge levels, or parts thereof, of which only the first two can be “*learned*” internally. We decided to keep the maximum knowledge of 100 (which is assigned by the unary Knowledge Update relation upon visiting a suitable page) to set the maximum amount of knowledge obtainable through the AHS (2nd knowledge level). Now, to emulate the InterBook outcome relation we used the formulas published in [6], to increase the knowledge of outcome concept O by $\frac{1}{3} * (100 - O.\text{knowledge})$ if the currently visited page is suitable (all prerequisites satisfied), or by $\frac{1}{6} * (100 - O.\text{knowledge})$ otherwise. This second formula is used to imitate the InterBook behavior that visiting a non-suitable page three times still accredits a full level of knowledge. Subsequent visits to that same non-suitable page should not further increase the knowledge level. In the AHA! outcome relation we ensured this by using the integer value of the visited attribute from the source (page)concept to store the percentage of the maximum amount of added knowledge that is accredited thus far, as long as the source concept is non-suitable. By using this approach the concepts “remember” for themselves how much knowledge they already have contributed (to all outcome concepts connected to them), and what they still can contribute in sub sequential visits.

The following is a fragment from the AHA! concept relation xml file that defines the above mentioned behavior, the first part of the fragment defines the behavior of the relation when the currently visited page (the source) is suitable (and has been visited before), the second the behavior for when the page is visited while it is non-suitable for the first time, as is coded in the “requirement” statements. The action elements define which attribute gets updated, and by how much:

```
<generateListItem location="child.access">
  <requirement>
    source.suitability & amp; & amp;
    source.visited > 0 & amp; & amp;
    destination.knowledge < 100
  </requirement>
  <trueActions>
    <action>
      <conceptName>
        destination
      </conceptName>
      <attributeName>
        knowledge
      </attributeName>
      <expression>
```

```

        destination.knowledge
        + 1/3*(100-destination.knowledge)
    </expression>
</action>
<action>
    <conceptName>
        source
    </conceptName>
    <attributeName>
        visited
    </attributeName>
    <expression>
        100
    </expression>
</action>
</trueActions>
</generateListItem>
:
<generateListItem location="child.access">
<requirement>
    !source.suitability & & &
    source.visited < 32 & & &
    destination.knowledge < 100
</requirement>
<trueActions>
<action>
    <conceptName>
        destination
    </conceptName>
    <attributeName>
        knowledge
    </attributeName>
    <expression>
        destination.knowledge
        + 1/6*(100 - destination.knowledge)
    </expression>
</action>
<action>
    <conceptName>
        source
    </conceptName>
    <attributeName>
        visited
    </attributeName>
    <expression>
        33
    </expression>
</action>
</trueActions>
</generateListItem>

```

One case that is omitted from the above fragment is the initial case when a concept is visited while it is suitable. In this case the full knowledge level is always added (with a maximum of the below third knowledge level). This is done because in InterBook visiting two distinct (suitable, non-visited) pages leads to an acquiring two knowledge levels.

Two other cases omitted are the coding for sub sequential visits to the concept while it is non-suitable. The only difference with the above fragment is a different limitation on the `source.visited` attribute, and the assigned value to the same attribute. Ideally we would combine this in a single case, with an expression `(source.visited + 1/3*100)`, but then for every outcome relation, the `source.visited` attribute gets increased by 33.

3.3 Look and Feel

An AHS is more than its concept structure (and its user model update and adaptation rules). Even when the adaptation behavior of a translated system is emulated perfectly, users will still have a very different experience if in the target system the layout of windows and frames is different, the background and colors on the pages are different, the link colors and annotations are different, etc. All these aspects together form the *look and feel* of the AHS.

In order to be able to “translate” the look and feel to the target AHS we have to consider at least the following user interface aspects of an AHS:

- The overall *layout* of an application consists of windows and frames (in HTML terms); certain actions might open a new window, while others cause some form of navigation using the “current” window.
- A window is often divided into *frames*. Typically one of the frames will display a page (with the main content, for the user to read) while other frames show additional information, such as a menu to be used for navigation, or information about the user’s knowledge. It is unlikely that the target AHS has exactly the right “views” to completely emulate the source system. One solution is to generate all the possible instances of these views in the translation and include the correct instance. Another solution is to write code to generate the views on the target.
- *Link adaptation* can consist of link hiding, link removal, or link annotation. (See [9] for an overview of link adaptation techniques.) Any of these adaptations needs to be recreated on the target AHS. Link color, link underlining and annotations (icons) are often typical for applications, and are usually based on the value of user model attributes. They can be based on the knowledge of a concept but also on the “suitability” (which is for instance a condition on knowledge of background concepts).

AHA! has always allowed the creation of arbitrary layout and presentation, but the authors had to create the frame structures and write JavaScript code to ensure that when following a link the other visible frames were updated as well. In version 3.0 AHA! was extended with a layout handler and view generator, making it easier to create applications that use windows and frames. Also defined in the layout model is the color and (links to) icons used for annotation. Finally, using the conditional inclusion of fragments, AHA! is able to support content adaptation (including link removal).

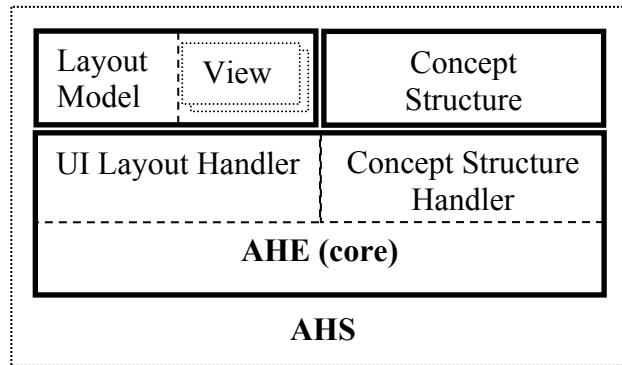


Figure 8. Internal structure of an AHS with AHE

Figure 8 shows the AHS architecture as used by AHA!. The adaptive hypermedia engine (AHE) consists of two main parts, one that handles the layout, based on a layout model and dynamically generated views, and a concept structure handler that deals with the user model updates and adaptation, based on the concept structure.

3.3.1 InterBook

As indicated before, in general an AHS is the combination of its concept structure and its user interface (UI). The rich InterBook UI is closely connected to the InterBook concept structure: we find the same separation between concepts and contents. InterBook presentations consist of two main windows, a glossary displaying information concerning the concepts (providing navigation support through the domain model based upon the concept relations), and a the textbook window concentrating on classical navigation through the contents.

The glossary is an important part for any (InterBook) ET. It should be considered as the visualized (and externalized) domain network. Each node of the domain network is represented by a glossary entry and each glossary entry corresponds to one of the domain concepts. The links between domain model concepts constitute navigation paths between glossary entries. Thus, the structure of the glossary resembles the pedagogic structure of the domain knowledge. In addition to concept description, in InterBook, each glossary entry provides links to all pages that introduce the concept. This means that an InterBook glossary integrates features of a traditional index and a glossary.

For each concept the glossary window presents a concept description (if available) and two lists of links to content pages (Figure 9). The first list links to pages that have the current concept as outcome concept (through any relation) and the second list links content pages that require knowledge about this concept (have it as prerequisite). This allows the user to explore and amend his knowledge on his own. This window also includes two concept selection frames.

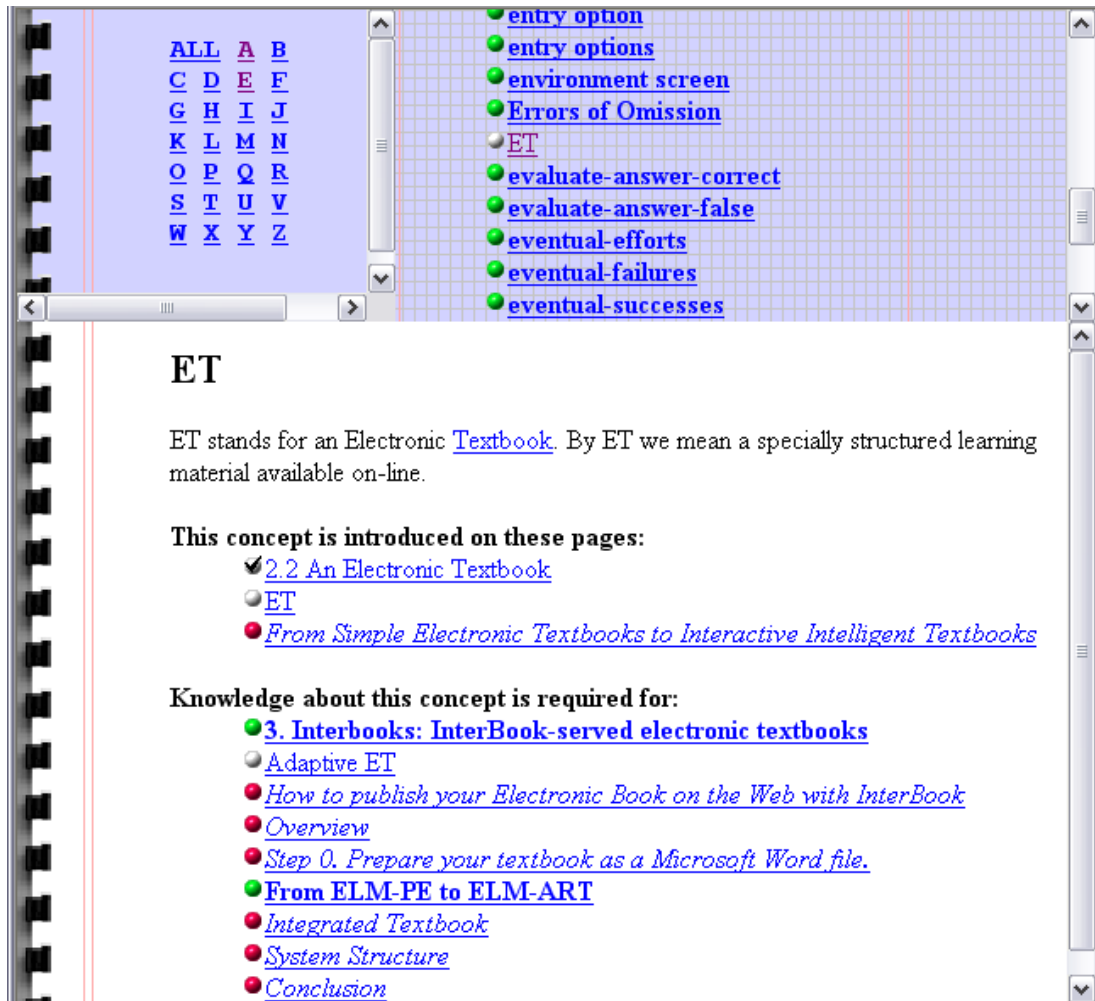


Figure 9. InterBook Glossary Window

The textbook window (Figure 10) has a main frame (*text window*, #3) that presents the content of a page, with a colored bar on the top to indicate the suitability, and links to the glossary from hot words within the text. If the page has children (e.g. if the page is a section that has subsections), then below the content text links to these children pages are provided. Two buttons (*Back* and *Continue*) allow for sequential browsing through the ET. Above the main frame the user is provided with a hierarchical *navigation bar* (#1, to the siblings, parents and grandparents of the page). This provides the user with a way to browse the ET in a more classical fashion. Then, to the right of the main frame the user is offered two lists (*concept bar*, #4), displaying prerequisite concepts (listed under “background”) and outcome concepts (from any type of outcome relation), with each concept possibly followed by one of three sizes of checkmarks to indicate the level of knowledge the user has of that concept.

Other minor views available through the *toolbox* area (#2) besides the Glossary are a *help view*, which will display a dynamically generated list of links to content pages that provide more information about the prerequisite concepts of the current content page, a *search window*, which allows the user to search in the active textbook, and a general *table-of-content* view. The functionality of the *Back* and *Continue* buttons is replicated in this area by the two buttons with arrows.

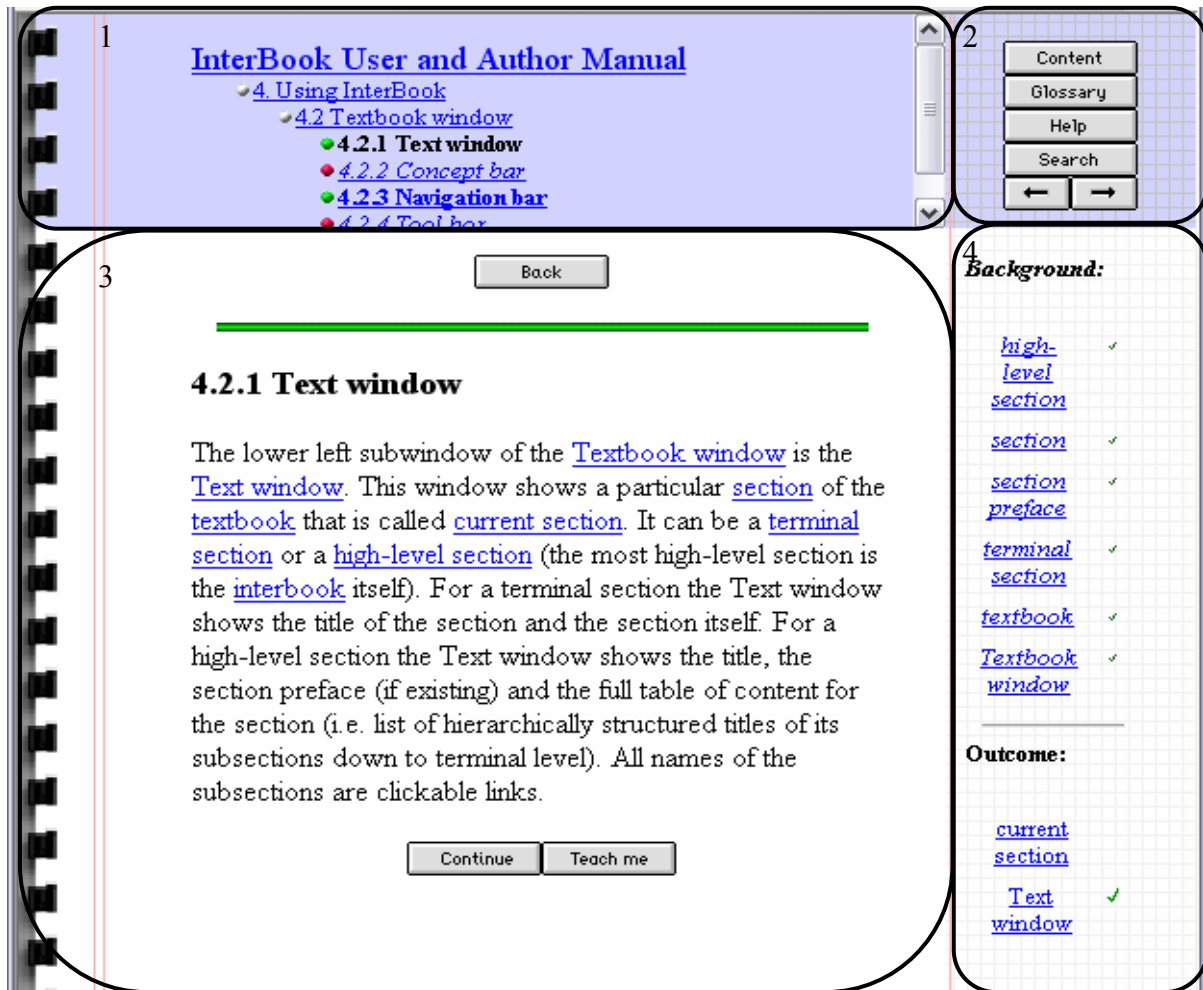


Figure 10. InterBook Textbook Window

3.3.2 AHA!

Although the AHA UI (as used in the AHA! tutorial) is of little influence to our translation because of the AHA! Layout Handler, we want to give a short description of it for completeness. Figure 11 displays a screenshot from the AHA! tutorial; the numbers indicate similar frames compared to Figure 10.

AHA! uses a single window, divided into two frames. #1 gives a hierarchical *navigation bar*, showing the siblings, parent and grandparents of the current page, plus their siblings. #3 is the *text window* with the main area (below the horizontal ruler), displaying the contents, preceded by an author-defined header, which can display a variety of information and links. Not visible is the author-defined footer, containing a copyright statement. Both footer and header are (optionally) automatically included in every page.

Links in AHA! are only annotated by color. *Red* links indicate a link to pages not part of the application, *blue* to application pages suitable for the user, *purple* indicates a link to an already visited page, and *black* links (e.g. "form editor") indicates a non-suitable page. Blue and purple are chosen because of their standard usage on the Internet, black because it effectively implements *link hiding* for non-suitable links in the contents.

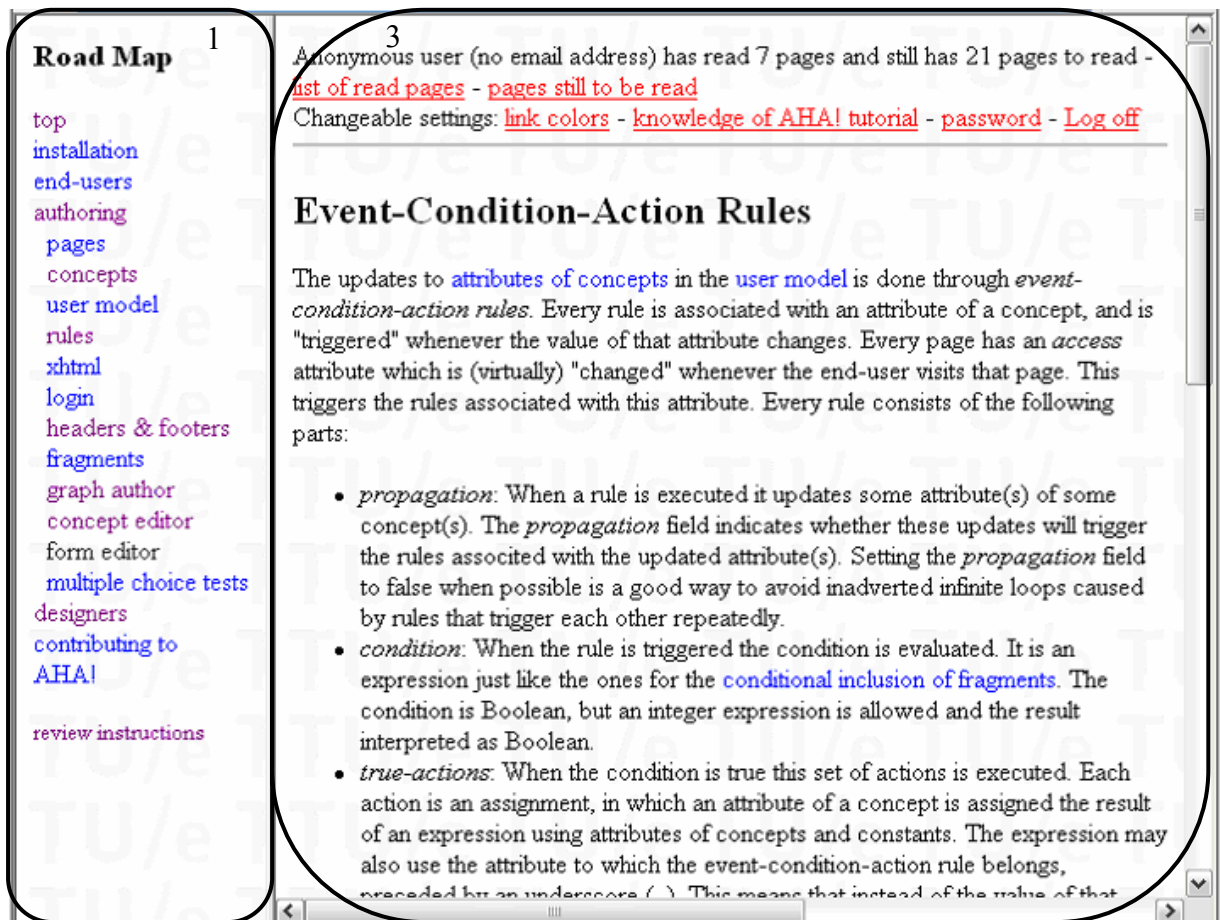


Figure 11. Original AHA! Application

3.3.3 latA

Possibly being one of the most influential factors toward the user experience when using an AHS, it is important to keep the UI from a translated AHS as close to the original as possible. Using several (partially previously built) AHA! views, we recreated the InterBook UI on AHA!. Figure 12 shows the new version of the Textbook window displayed in Figure 10. This layout is realized through the following definition in the application specific Layout Config file:

```
<viewgroup name="InterBook Textwindow">
  <compound framestruct="rows=20%,*" border="0">
    <compound framestruct="cols=*,175" border="0">
      <viewref name="v1" />
      <viewref name="v2" />
    </compound>
    <compound framestruct="cols=*,175" >
      <viewref name="v0" />
      <viewref name="v3" />
      <viewref name="v4" />
    </compound>
  </compound>
</viewgroup>
```

“v#” refer to a view declaration¹, e.g. v1 refers to the following declaration (from the same file):

```
<view name="v1" type="TreeView" title="Tree view"
background="icons/pathview.gif" params="leftspace=10">
```

Indicating that the (in Java defined) TreeView class type should be used in that area, with “icons/pathview.gif” as background, and the text preceded by a 10 pixel wide blank area.

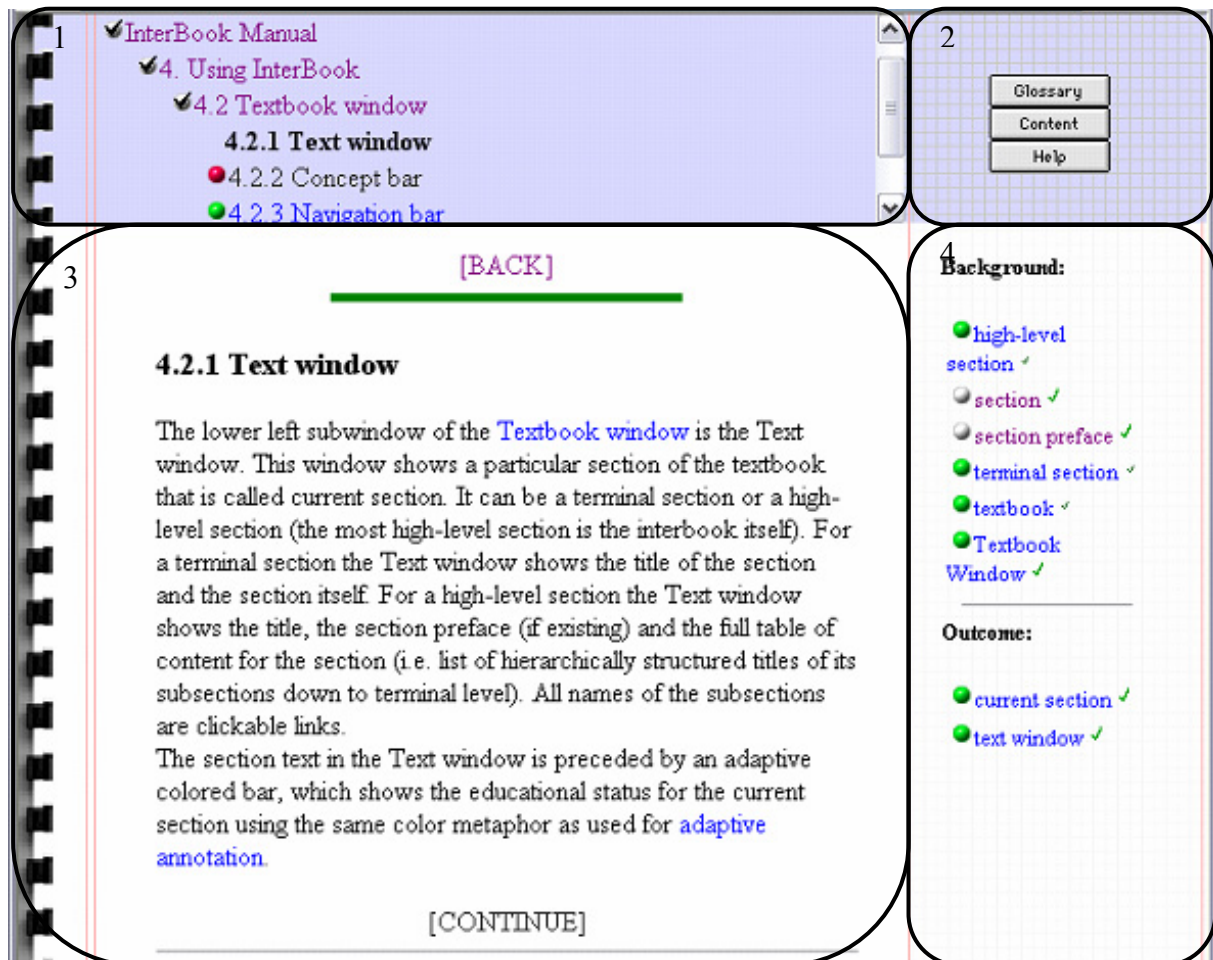


Figure 12. InterBook after Translation

The other views and the Glossary window (Figure 13) are defined in a similar fashion. The difference between Figure 13 and Figure 9 is mainly due to listing the two lists next to each other instead of underneath each other.

¹ v0 refers to an “empty view” with no contents besides a background, This is used to avoid duplication of the background image in the main text area on large screens. This approach is possible because of the white background behind the text. The Navigation Bar (#1) has the same problem on wide screens, but because it would still require a background behind the text, and a re-occurrence of the left side of the image is unlikely to interfere with the text, we have opted out for this.

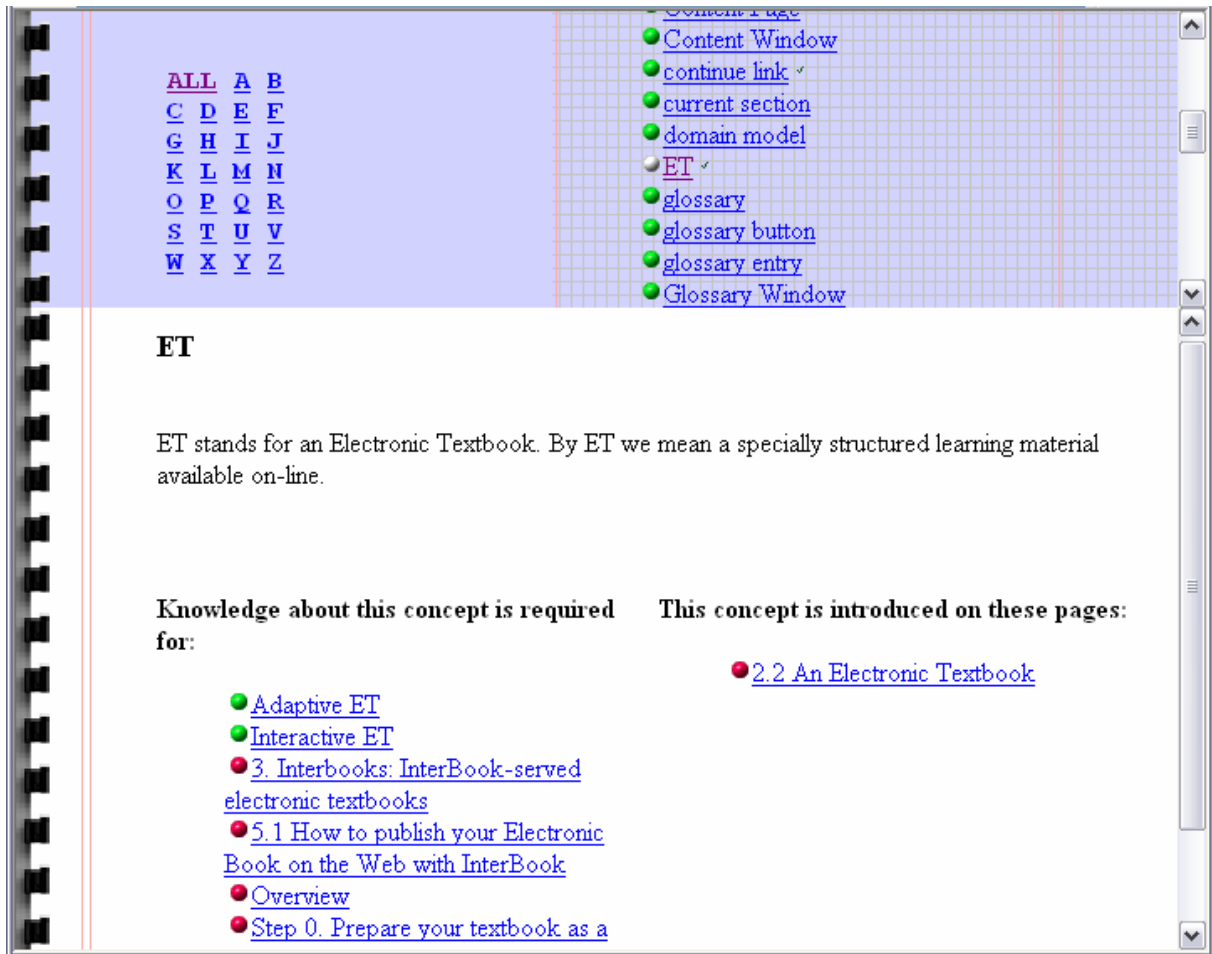


Figure 13. The AHA! Glossary

3.3.3.1 Help: Adaptive Sorting in AHA!

InterBook has the ability to support the user to understand a non-suitable page if the user visits that page anyway, through a help window. This window will list all pages that have the prerequisite concepts (for the current, non-suitable page) as outcome. The links are sorted in two ways; first, they are grouped in suitable, non-visited pages, visited pages and non-suitable pages. Within these groups, the links are further sorted on the number of prerequisite concepts from the current page that they have as outcome concepts. In Figure 14, section 2.1 is listed before section 1. This means that the user will learn more necessary concepts by reading section 2.1, than when reading section 1.

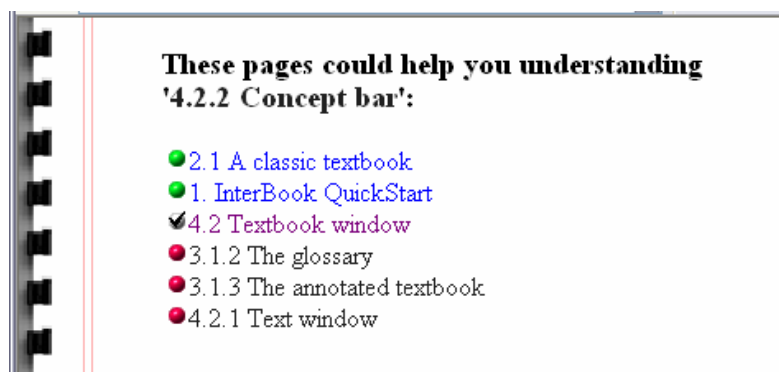


Figure 14. AHA! Help View

The used sorting algorithm for the AHA! Implementation is the Bin-sort with additional knowledge. Using this additional knowledge, it is possible to limit the number of bins (to the number of prerequisite concepts for the current page), and keep the complexity On where n is the number of concepts within the application².

3.4 Differences with InterBook

When we inspect InterBook and IatA closely, several difference will be noticed. A very careful inspection will even reveal a difference in functionality. Some of the differences are by choice, some because of certain limitations. We will shortly discuss these differences here.

A difference in functionality, by choice, is a different functionality for the *exemplifies* and the *introduces* relations. Although they are based upon the standard outcome relation, we have added certain restrictions to them, to research a more optimal usage of these relations. The *introduces* relation has the added limitation that it will not increase knowledge of the destination concept beyond the first level. The idea behind this is that if a concept is only “introduced” in a piece of text, it does add some knowledge to the user, but because it is not entirely explained to the user, it should not be able to increase the user’s knowledge too much.

The *exemplifies relation* is complementary to the *introduces* relation. Because an example only makes sense when certain knowledge about the concept is already present, we have it require the first level of knowledge to be present, before it will add knowledge to that concept. Besides the straightforward usage it could also be used to author conditional fragments (Section 4.3.2) based knowledge assignment. By also requiring the first level of knowledge before presenting the conditional fragment (with the example), the display of the example and the increase of knowledge would occur under the same conditions. Otherwise more advanced authoring is required to achieve similar functionality.

There is not only a difference in functionality; there are also several differences in the UI. Comparing the Textbook windows in Figure 10 and Figure 12 we can observe the following differences:

- Within the text frame, InterBook will automatically generate links from occurrences of (global) concept names. AHA! does not have this functionality, and because through manual authoring these links can also be created (Section 4.2.2.1) we gave precedence to other functionality.
- Not because of precedence, but because of technical limitations, is the omission of a search view. InterBook has the option for users to search the contents of an ET. However, because AHA! has conditional contents, an occurrence in the source contents might not be displayed when the user would actually visit the page. The influence of external information (user’s knowledge) causes this search to become non-trivial.
- Both because of precedence and possible technical limitations, there is no “Teach me” button in IatA. To exactly calculate which page to read to obtain prerequisite knowledge is non-trivial because of the conditional fragments (and the conditional assignment of knowledge by the *exemplifies* relation). The implementation of the Help window is different: this is a recommender system, and it does not guarantee that the topmost listed page is indeed the most appropriate.

² This might not be entirely correct. To calculate the bin number to which a page belongs, we calculate the size of the intersection between its set of outcome concepts and the set of prerequisite concepts. Mentioned complexity expects a linear complexity for the JAVA implementation of `linkedList.retainAll` and `linkedList.removeAll`.

- The AHA! toolbox window is implemented as a portal, only providing links to secondary windows. In InterBook, two additional buttons are added, with the same functionality as the *Back* and *Continue* buttons in the main text window. To maintain the simplicity of the AHA! toolbox view, and avoid duplication of functionality, we decided to omit these two buttons from the toolbox window.
- The mentioned *Back* and *Continue* buttons are another difference themselves. In AHA! we use textual links instead of buttons. We chose for this difference because it allows (color) annotation of both links with the AHA! color scheme. In Figure 12 the previous page has been visited, while the next page is still non-suitable.
- Additional annotation of links is also one of the reasons for the choice to display bullets in front of the concept names in the toolbox window. They inform the user about the suitability of concepts. Although rare, a concept can have prerequisite concepts too. The other reason is the continuity of annotation. In InterBook links to concepts from within the Glossary do, but links from the toolbox window do not have the bullet annotation.
- Murray [20] researched the influence of depth-first versus breadth-first navigation through hyperbooks, and the best style of hierarchical navigation support. Based upon this research we decided to differ from the InterBook approach, and list the children of a page on top together with the siblings and (grand) parents, instead of listing them separately below the contents.

Finally, in an attempt to improve the navigation through the Glossary, we tried to order the links in a better way, continuing on previous InterBook research. Although the original goal was to sort them on relation type (outcome, introduces, exemplify), because the link types are not known in runtime, we decided to sort the links on origin and direction. In a 2 by 2 table we have separated links for:

- Pages (and concepts) that have the currently displayed concept as outcome;
- Pages (and concepts) that have the currently displayed concept as prerequisite;
- (Pages and) concepts that are prerequisites to the currently displayed concept;
- (Pages and) concepts that are outcomes from the currently displayed concept.

4 Authoring

As we stated in [22], it is not enough to just place a textbook online, with links to jump quickly to chapters and sections, because it can result in a frustrating experience of non-understanding to the adventurous learner. However, this is often what happens, instead of using adaptive hypermedia techniques to create a more adaptive version. We think the main problem is the lack of authoring tools to support the creation of applications for AHS.

AHS have the advantage that the user is guided through the content in some way, but if no applications are available for it, there is no use for it. Applications consist out of two parts: *contents*, which is information displayed to the user, and a *domain model*, organizing the contents in some fashion. Creating these applications (authoring) can be supported in many different ways, and can focus on *one* or *both* of *contents creation* and *domain model* (or *concept structure*) *creation*.

Both AHA! and InterBook focus on one of these two areas. AHA! has extensive support for authoring the domain model, but requires manually creation of contents through HTML editors (Section 4.1). InterBook facilitates authors with the ability to easily create a standard textbook which is then transformed into an application, but (the annotation with) the domain

model is done manually, while the author needs to have a mental idea about the structure. (Section 4.2)

Our second goal was to unite the ease of contents creation from InterBook with the powerful concept structure creation from AHA!, obtaining one system with support for both areas, because authors prefer to use a word processor to create a textbook over an HTML editor to create lots of individual (content) pages.

4.1 AHA!

Since AHA! version 2.0 a graphical tool has been available that lets authors draw the structure of concepts and concept relationships for an ET. This tool is described in [7]. So unlike in InterBook where the concepts and concept relationships are defined as hidden constructs that are scattered throughout the textbook this structure is represented by a graph and visualized and edited using the Graph Author tool.

The content pages of an AHA! application are created using any HTML editor. Each page must have a corresponding concept in the concept graph. Since AHA! version 3.0 links can not only refer to other pages but also to concepts. However, these links need to be created manually through the HTML editor.

AHA! has a very flexible presentation and layout module, first introduced in [13]. It allows for a different layout for different types of concepts (like for pages and for glossary items), consisting of frames with menus giving access to the hierarchical structure of chapters, sections and paragraphs, prerequisite concepts, outcome concepts, etc. It allows an author to choose different link annotations through colors as well as icons, etc. Every aspect of the presentation, like which frames exist, where which frames appear, what backgrounds and buttons to use, which link colors, which link annotation items, is defined through a layout specification. The presentation style of many different adaptive hypermedia systems can be emulated in AHA! (see Section 3.3.3 for a discussion on the layout specification used for realizing IatA). All these specification are (manually) configured in two XML files.

4.2 InterBook

InterBook (Brusilovsky et al, 1998) is designed to help a course-author to transfer a normal textbook existing in electronic form into an adaptive electronic textbook. Therefore, it focuses highly on contents and its structure, while the annotation is designed to allow for normal printing also after annotating. The conceptual structure needs to be realized mentally and manually. However, once the structured content is (manually) indexed by concepts (and concept relationships) it can be automatically transferred into InterBook structures.

It works on textbooks in RTF format, as produced by e.g. Microsoft Word, or any other Word processor that can create the required markup. In order to use InterBook, the textbook must be structured hierarchically using “header” tags to indicate the start of chapters and sections. (Simply creating a line of text with a large bold font does not produce a recognizable section or chapter title.) We will shortly discuss the procedure of transferring a normal textbook to InterBook next, because most of this process is maintained in our translation.

4.2.1 Publishing Electronic Textbooks on the Web with InterBook

To publish an ET on the Internet with InterBook, the author needs to go through 5 steps which are described below (see also Figure 15). In brief, the author must prepare the ET as a specially structured RTF file and then convert this file into InterBook format. The result of

this process is a file with the Textbook in InterBook format which can be published on the Internet by the InterBook system.

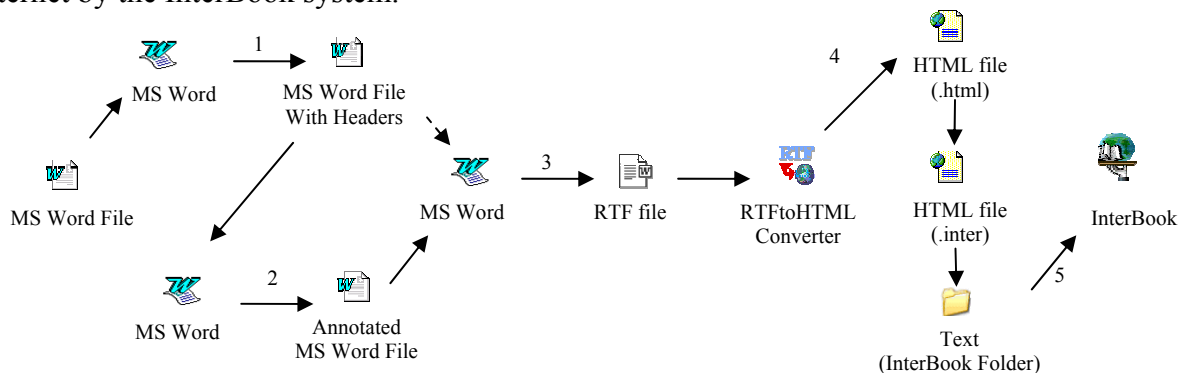


Figure 15. Serving an Electronic Textbook on the WWW with InterBook

1. *Structuring of the RTF file.* To let InterBook recognize the structure of the ET, the titles of the chapters should have the pre-defined paragraph style “Header 1”, the titles of sections should be “Header 2”, and so forth. The title of the textbook should have paragraph style “Title”. This structure is necessary because InterBook’s RTFtoHTML converter will use these section headers to decide how to divide the textbook into pages. The Word file can be converted to InterBook format straight away via step 3, but no adaptive features will be present.
2. *Annotating the MS Word file.* The concept-based annotation of the ET is needed to let InterBook know which concepts correspond to which section. This knowledge lets InterBook help the reader of the ET through adaptive navigation support. Although it is not necessary to design the structure of concepts before writing the paragraphs and sections of text, most authors will start with an outline and structure of a textbook before writing anyway. This step could be skipped, but it is generally not recommended. However, because for e.g. testing purposes it might be useful, it is indicated in Figure 15 with the dashed arrow.

An annotation is a piece of text in a special style and format, inserted at the beginning of each section (between the section header and the first paragraph). Annotations have the special character style (*hidden*³ + *shadowed*). For each (sub)section, the author can provide an (optional) set of *outcome* and *prerequisite* (or background) *concepts*. The format for the *outcome annotation* is:

(out: concept-name1, concept-name2, etc.)

The format for the *prerequisite annotation* is:

(pre: concept-name1, concept-name2, etc.).

3. *Conversion to RTF format.* If the text book is not in RTF format yet, it needs to be saved in RTF format. The used conversion program (RTFtoHTML) takes RTF formatted files as input.
4. *Translating from RTF to (InterBook-)HTML.* The RTF file with the ET is translated into an HTML file by the RTFtoHTML program controlled by some specially designed settings. In particular, all annotations are translated into HTML comments with a special format. RTFtoHTML will extract the images and create a single HTML file with the same name as the RTF file, but with the .html extension. This extension needs to be renamed to .inter and the file needs to be transferred to a special “Text” folder on the InterBook server.

³ Note that it may be necessary to turn-on the viewing of formatting marks for *hidden text*

5. *Parsing into LISP structure and Serving on WWW.* When the InterBook server starts, it automatically parses all interbook files in its “Texts” folder (i.e. all files with extension “.inter”) and translates them into the list of section frames. Each section frame contains the name and type of the unit, its spectrum, and its position in the original HTML file. The obtained LISP structure is used by InterBook to serve all the available textbooks on WWW along with providing all advanced navigation and adaptation features. All content which the user will see on the screen is generated on-the-fly with the knowledge about the textbook, the user model, and HTML fragments extracted from the original HTML file. These features of InterBook are based on the functionality of the Common Lisp Hypermedia Server CL-HTTP.

4.2.2 Advanced Authoring for InterBook

Besides translating a textbook directly/literally to an ET with added adaptive annotation, it is possible to add even more power to the ET. Hypermedia allows for creating links from virtually anywhere to virtually anywhere. Crosslinking between sections, linking to concepts in the Glossary, and linking to external (Internet) content is also possible within InterBook, author-able directly from the RTF file. Additional descriptions to certain concepts can also be created from the RTF file. Finally, it is possible to Bypass the automated content creation, and completely create the .inter files manually.

4.2.2.1 Creating Links to Internal Concepts and Section

InterBook can handle *links* to *concepts*, *sections* and *arbitrary URLs*. They are not created as Microsoft Word links for historical reasons. A link to a *concept* or *section* consists of two parts. The first part is the name of a *concept*, *section*, or the original URL. It has to be formatted as a *hidden* and *double-underlined* text. The second part immediately following the URL is the *hot word* or *hot phrase* (the link anchor). This phrase has to be formatted with the character styles *single underlined* and *italic*. RTFtoHTML uses this formatting to generate a proper link. Because the link anchor is not hidden, any word or phrase that appears in the ET can be turned into a link anchor.

4.2.2.2 The Glossary

As mentioned before, the Glossary is of large importance to interbooks. It provides the user with access to the pedagogical structure (which might be very different from the original textbook chapter structure) in the role of *index*. However, the second role is where it obtained its name from, Glossary. It has the option to display a *definition*, or *concept description*, to the user.

In the Word file, the concept description should be added at the end of the file, with “Glossary” (Heading 1 paragraph style) indicating the start of the glossary, followed by the concepts with their concept description (with the concepts in Heading 2 paragraph style). It is possible through regular annotation to add (knowledge about) other concepts as prerequisite in the same way as to content sections.

4.2.2.3 Editing HTML files.

An important fact is that RTF files and RTFtoHTML are only required to make authoring easier. What InterBook really uses is an annotated HTML file. An experienced author can bypass step 1, 2 and 3 by preparing the textbook directly in HTML format with annotations provided as specially formatted comments. Direct HTML editing can also be useful if you can not create a particular HTML effect using RTFtoHTML. The rules are very similar to the

rules of creating an .rtf file. You have to add two types of meta-information to your .html file: the structure, and the concept index.

To tell InterBook about the structure, place a special comment

```
<!--#n#-->
```

before each html <Hn> header where n represent the level of your header. The title of a book has to be prefaced by n=0. To tell InterBook about concepts indexing a section, place a special comment

```
<!--((pre: concept1)(out: concept2, concept3))-->
```

after each section's header.

4.3 (Extended) IatA

In Section 4.2 we described how a Textbook could be transferred from RTF file to InterBook. This content creation has the advantage that an author can first create the content, possibly while creating the contents for a hardcopy version, and then continue by annotating that content to allow for the translation onto InterBook. Because this approach is generally preferred over an HTML editor, especially when creating a large application (with many content pages), we tried to maintain as much as possible from this process, while having the final output in AHA! format.

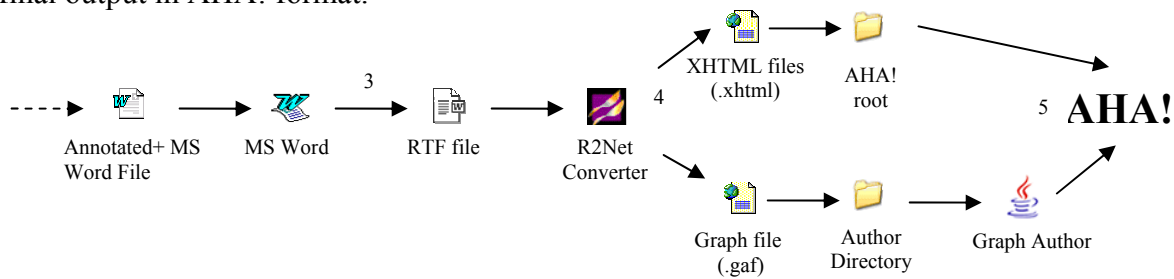


Figure 16. Serving an Electronic Textbook on the WWW via InterBook to AHA! translation

We decided to replace step 4 and 5 from the InterBook process (see Figure 15 and Figure 16). This allows us to use the same (RTF) input files. RTF2HTML has evolved in the more powerful R2Net (see also Appendix A). We were able to adapt the script in such a way that it produces both types of files required by AHA!; the XHTML files with the content, and a graph file with the concept structure, for input to the Graph Author. The Graph Author replaces the InterBook step 5, to publish the application to the AHS, and add the application (and its concepts) to the AHA! database. Here the author can also see a graphical representation of the created concept structure, although, especially when many (page and global) concepts are involved, this might not be too enlightening.

We decided against using the approach used in the earlier attempt [6], which branched off after step 5, using the .inter files as input for yet another process, because of two reasons. Firstly, we were aiming at simplicity, adding an additional step/process would only complicate the authoring process. Secondly, we intended to extend the translation process with new AHA! specific annotations. Branching off after step 5 meant that we would have to invent an additional sub-stage annotation to add to the .inter files. By branching off already after step 3, we avoid this.

After we had successfully created this translation, which supports all annotations from InterBook, we continued by extending the annotation (and translation process) to support AHA! specific features as like the *conditional inclusion of fragments*. This imposed one

limitation on the support of InterBook annotations, but because an alternative was possible (see Section 4.3.4), we have accepted this limitation.

4.3.1 New Concept Relations

One of the last research topics on InterBook were new relations. These *introduction*, *example*, *is-a* and *part-of* relations existed but did not yet have any special functionality. We decided to also support these relations in our translation, even though the exact functionality is still under study. That poses no problem however, because their inclusion in the *conceptual structure* and the translation from the RTF file to the AHA! format is independent from the (low level rules defining the) *adaptive behavior* for these relations (which is expressed elsewhere in AHA!'s concept relationship templates, see also Section 3.4). It allows us to support the relations in the translation, even though changes might still be made to their functionality in a later stage.

The *annotation format* is very similar to that of outcome and prerequisite. In *hidden* and *shadowed* character style, at the beginning of the section (same as for outcome and prerequisite), the author needs to include

```
<relation>: concept-name1, concept-name2, etc.)
```

where <relation> is *intro* for the introduction of concepts, *exam* for concepts that are exemplified in the section, *isa* to indicate an is-a relation with other concepts, and *partof* to indicate a part-of relation with other concepts. The first two are generally used to annotate sections; the later two are usually used in the glossary to indicate relations between concepts. The currently implemented functionality is described in Section 3.4.

4.3.2 Conditional Content

In AHA! it is possible to use conditional fragments to (not) include some text in one page. In InterBook, this was not possible, and therefore there was no RTF markup available to support this. To be able to use this functionality from AHA! through RTF files too, we extended the RTF markup, thereby trying to stay close to the original InterBook idea, that it should be easy to use, also for novices, but also tried to include a maximum of the expressive power of AHA!. The compromise consists of using near-natural language and allowing a shortcut for expert users. The basic form is:

```
(if: <condition> ;<text>)
```

and should be in *hidden* and *shadowed* font style. <text> defines the text fragment that is conditionally displayed, and it can either be a *single* text block (to use when <condition> evaluates to true) or have an entry for *both* the true and false case of the condition, separated by a semicolon (;). Hence a semicolon can not be used within these text blocks.

<condition> is more complicated, it would be defined as follows, where *concept* can be any concept name:

```
<condition> =  
(not? {suitable|visited|very? well? known|#} concept)  
{and|or}  
(not? {suitable|visited|very? well? known|#} concept))*
```

Or in plain English, it consist of at least one attribute/concept pair, optionally preceded by “not”, optionally followed by a Boolean operator (“and” or “or”) and another attribute/concept pair (again with optional “not”), and with arbitrary (valid) placing of

parentheses. This can then again be followed by more Boolean operators and possibly negated attribute/concept pairs. The attributes translate to the AHA! concept attributes as follows:

- “suitable” and equals the “suitability” and attribute
- “visited” equals the “visited” attribute
- “very well known” translates to “knowledge>=95” (InterBook Knowledge Level 3)
- “well know” translates to “knowledge>=66” (InterBook Knowledge Level 2)
- “known” translates to “knowledge>=33” (InterBook Knowledge Level 1)
- “#” translates to “knowledge>=#”, with # being any integer number, allowing a shortcut to arbitrary values for more advanced users.

An example of a correctly formatted conditional fragment (except for the *hidden* and *shadowed* markup) would be:

```
(if: not known interbook system and suitable interbook
system; some text for people without (much) knowledge
about InterBook; something for InterBook Experts)
```

which translates to AHA! format as :

```
<if expr="
  !(interbook_manual.interbook_system.knowledge>=33) &a
  mp; &amp; (interbook_manual.interbook_system.suitability)
">
<block>
  some text for people without (much) knowledge about
  InterBook
</block>
<block>
  Something for InterBook Experts
</block>
```

Abovementioned *hidden* font style causes that nothing from the conditional contents shows up if the (original) textbook is printed. This allows for an equal treatment of true- and false-fragment, while avoiding (possibly) strange text-flows caused by different wording for a similar text fragment when both cases would be shown. It is, however, not possible to add links or special markup (e.g. bold) or nested conditional content through the standard annotation to the fragments. This is because R2Net does not further process the content contained within the fragment. However, it is possible to manually add html tags. They will be maintained and carry over into the final content.

4.3.3 Reusable Contents

A new feature of AHA! 3.0 is the selective inclusion of reusable objects (through transclusion). It allows for the definition of multiple external fragments, each with their own condition, of which one is included when it is linked from a content page, based upon the conditions. Although it is a very powerful feature, it would require the transfer of a lot of information via the RTF file to the final application. An annotation which fully supports this feature would become a very complex annotation schema. However, having the conditional fragments, we decided to reduce selectively included reusable objects to simple reusable objects, because with a thoughtful use of the conditional fragments in combination with reusable objects, it would still be possible to produce a similar effect as selectively included reusable objects have.

A reusable object is a fragment of text that can easily be made to re-appear on several different content pages in the final application. The main markup here is a *hidden* and *shadowed* text fragment, in the form of:

```
(reuse:concept1)
```

where *concept1* is any concept name (similar to “(pre:concept1)” for prerequisites). The text fragment of the object that will be displayed, is either a *glossary entry* (in which case there should be a glossary entry with description for the concept), or it can be defined *inline* by using (reuse:concept2, def) in *hidden* and *shadowed* markup *before a paragraph*⁴. That whole paragraph will then be used as text fragment for that object. A concept name used in an inline declaration (like *concept2* above) should be a *new* concept. Any occurrence of (reuse:concept, def) *plus the following paragraph*, or (reuse:concept), will be replaced with the aha object tag:

```
<object name="interbook_manual.concept" type="aha/text">
```

which will be replaced with the relevant text fragment during the presentation of the final application.

4.3.4 Internal Links to Contents and Concepts

In Section 4.2.2.1, we described the required annotation format to create internal links (both to other sections and to concepts in the Glossary). However, to allow for the use of global concepts to be used as reusable content, we had to place the glossary description under “default_fragment” in the concept description. Before the extension we had placed it under “concept_resource”, which allowed AHA! to automatically detect the accompanying concept, and replace the link generated by R2Net (to the resource file) with a link to the concept. The change to default fragment prevented this technique from working, and broken links were the consequence, because R2Net would detect the header as name of the description in the Glossary part of the textbook, and automatically create a link to that resource file.

We solved the problem by using the MS Word ability to generate a crosslink (an internal link to any header). R2Net can detect these crosslinks, generate an HTML link to the required section (resource) too. Internally it treats both link generations (crosslinks and manually annotated links; see also Section 4.2.2.1) differently however. Therefore, we are able to treat all manually annotated links as links to concepts, while letting R2Net continue to create links to resources for crosslinks.

5 Actual Translations and Encountered Problems

The first goal of this project, to translate currently existing interbooks to AHA! applications resulted in three interbooks (the original InterBook Manual, an interbook on Databases, and an interbook on Act-R) actually being translated. However, in the process we encountered several problems, which we have roughly separated into two sections, graph and concept related problems, and other problems. Bugs within the AHA! code that were encountered will not be listed here, nor anywhere in this paper because both their existence as well as their fixes are irrelevant to this project.

The second goal, to obtain a reusable authoring-through-translation for easy content authoring in AHA! is currently being tried in the development of a C-programming application (see also Appendix B). So far this seems successful, but because this development has only just begun, concrete results are not known yet.

⁴ Inline declared text fragments will not show up in the glossary during the presentation of the final application.

5.1 Graph and Concept Related problems

The first to be translated was the InterBook Manual. Being (much) larger than the AHA! tutorial (with only 28 page concepts and 57 relations), we ran into the problem of organizing all (automatically generated) concepts in a sensible fashion when displayed in the graph author. The InterBook Manual generated 39 page concepts, 46 global concepts and over 150 relations. We decided upon a double organization of two rows of columns of 10 concepts each (see Figure 17).

The top row of columns contains only page concepts, the bottom row only global concepts. This organization allows for (some) sensible viewing of the graph. We tried to increase the distinction of the relations by making all prerequisite-like relations in the red (purple)-yellow spectrum, and the outcome-like relations in the blue-green spectrum.

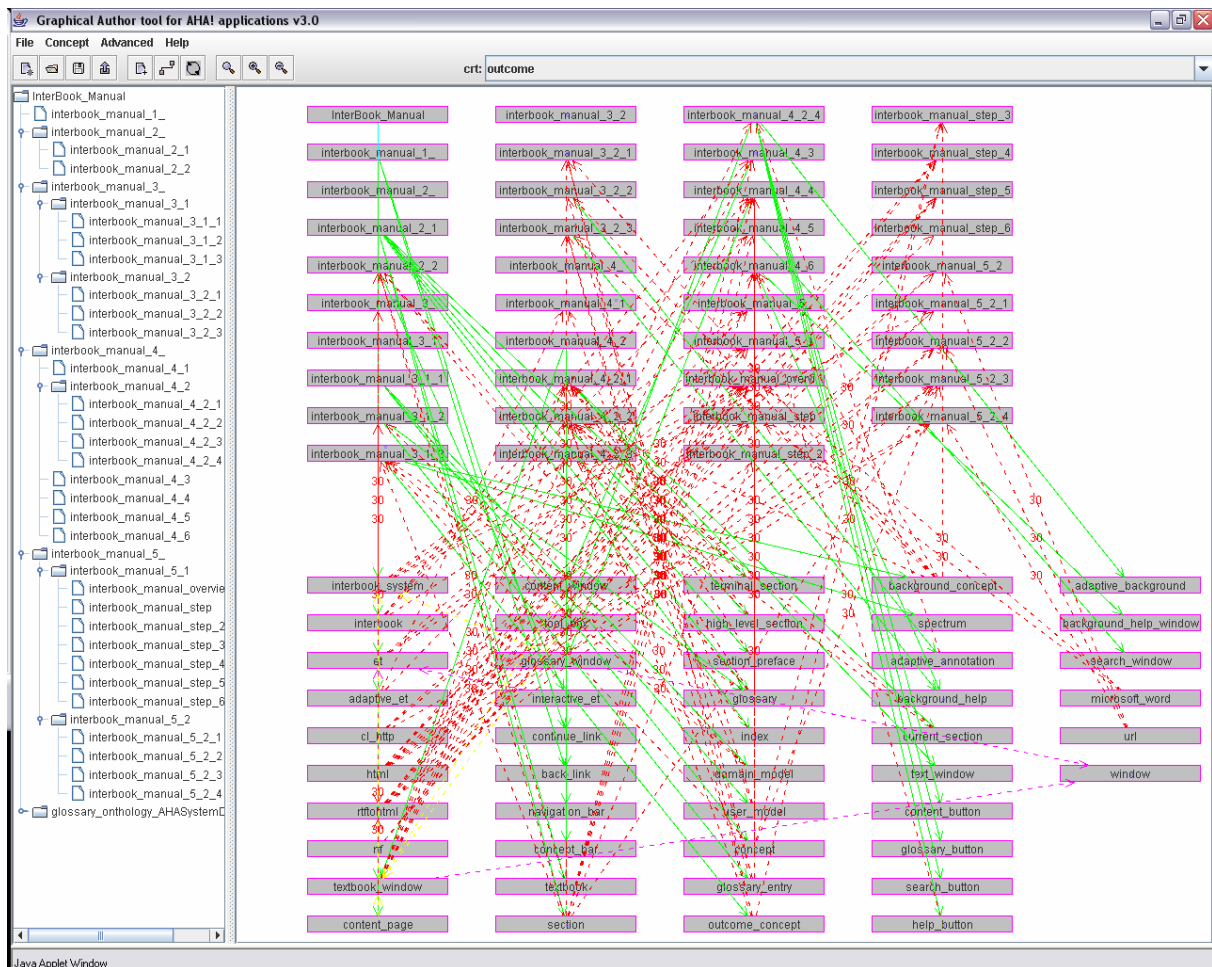


Figure 17. Graph for InterBook Manual

However, the limitations of the straightforward approach of the display of graphs by the graph author were even more obvious with the translation of the second interbook, on databases, and the third interbook, on ACT-R. This last interbook generated 155 page concepts, 209 global concepts, and over 650 relations. Although the separation of concepts types by color allows for some understanding of the graph when zoomed in, zooming out far enough to display the entire graph causes any meaning to be lost, and it resembled at most a work of modern art (Figure 18).

Another problem we ran into with all three files was errors in the annotation. Spelling errors in concept names, capitalization and the omission of the colon behind index words or the comma between concepts caused incorrectly generated graphs. The capitalization problem was resolved by making all global concept names lowercase. Spelling errors and omitted commas are considered the responsibility of the author; the translation will generate new global concepts in these cases. For our case we had to manually correct any such error in the rtf files.

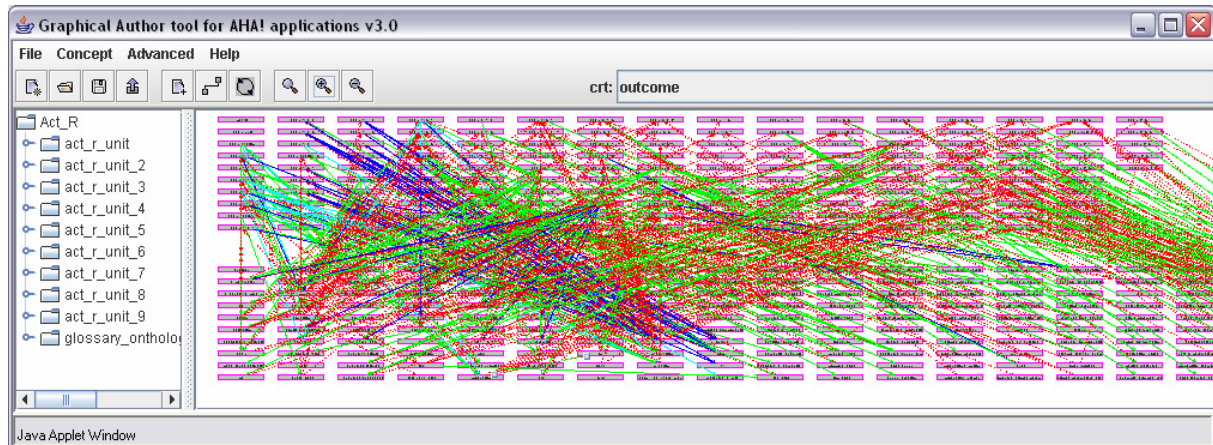


Figure 18. Act-R Graph

Interestingly, most of the before-mentioned concept name errors would also cause the original courses to misbehave within InterBook. Although the some errors were due to changes made after the conversion from RTF to InterBook (and were thus not present in the online courses), some errors indeed caused an incorrect knowledge assignment. Most errors present in actual InterBook courses were not critical, in that (multiple) other pages also would contribute knowledge to the involved concept, and therefore probably never noticed. However, in the Graph Author they often showed up as (global) concepts with only an incoming outcome relation, and a similar name as another (global) concept. Nevertheless, this way of troubleshooting through the Graph Author is a very elaborate task for larger graphs (Figure 18), especially because it is not uncommon that certain global concepts are not used as prerequisite concept within the application.

The concept names also gave rise to another problem; InterBook allows for any concept name, and there was no limitation on characters. AHA! does not allow certain characters in concepts names, e.g. the exclamation mark and the dot have special meanings within AHA! and therefore are replaced by underscores.⁵

5.2 Other Problems

Besides problems related to the graph generation some other problems were also encountered. Although they often were of less influence, we do want to list them here. One such problem was actually the working of InterBook. It is possible for InterBook to use multiple input files for one ET. These files are generated by separate RTF files. Because the AHA! graph author requires a single .gaf-file we needed to concatenate these separate rtf files into one before conversion. Adding information to existing files was not possible with R2Net, and we

⁵ Ironically enough, AHA!'s own name can not be used as concept name either, because of the exclamation mark at the end.

preferred a simpler translations process, so we did not add an extra phase after the translation to merge files. The concatenation of several files can be easily done within MS Word.

Although we stated before that any word processor can be used that can output RTF files, with the additional requirement that it should be able to produce the required markup, we inherited this claim from InterBook. The limitation of this claim became clear when we attempted to use Open Office. The “hidden” character markup is not available in this application, and it can thus not be used.

A final problem with the InterBook authoring process and the size of the produced graphs is independent of the graph author, and thus we did not list it there. It is the problem of the validity of the created application. Because of the complex event-condition-action rules it is possible that certain pages will not become suitable through any possible path. This problem is inherited from InterBook and as such left without further research of a solution. It is left to the author to fully test the application after submission to AHA!. Using the TOC, or the Continue button, and the conceptbar view it is possible to identify concepts for which no knowledge is obtained (in time). Using the Glossary it is then possible to identify whether there is no page with the particular concept as outcome, or that the page is introduced after the problematic page, not contributing its knowledge in time.

6 Remaining and Future Work

Although a lot is achieved in this project, there is definitely work left, which should still be done to optimize the resemblance between InterBook and IatA, would improve the usability of the Authoring-Through-Translation process, or would be an improvement over InterBook (and therefore IatA) in general. In the following sections we will shortly introduce some of that work.

6.1 Optimizing Resemblance

In InterBook, it is possible to obtain knowledge by visiting a concept in the Glossary, but only if a description of the concept is present; visiting a concept without description will only provide the user with links, accrediting knowledge for viewing those would be incorrect. However, in the current implementation of Global concepts (and the unary outcome relation) do just that. It does not distinguish between Global concepts with and without attached description. It always increases the knowledge to a maximum of the first knowledge level (AHA! value 33). To increase the similarity of functionality this differentiation should be made.

Another point on which IatA does not follow InterBook fully yet, and which is part of the user experience, is the annotation of links to pages. In InterBook, if a page has no outcome concepts, a link to the page is annotated by a white (neutral) bullet, also when not visited. Because in AHA! all pages have always an outcome (itself), internal mechanism and previous development of the hierarchical views do not support this additional page-state (neutral-unvisited). In AHA! a page only turns neutral when it is visited; only then will it not increase any knowledge anymore.

One aspect of InterBook we have not tried to translate yet is that InterBook allows the grouping of interbooks on a single Bookshelf. This means that although the user accesses one interbook, the (global) concepts contained in the Glossary are the superset of all (global) concepts contained in all interbooks from the active Bookshelf. Knowledge about concepts is also shared between the interbooks, as well as that it is possible to surf to other interbooks

through the Glossary. A (very limited) approximation to this in AHA! is to combine all separate interbooks into a single RTF file, and create one application. Here, all concepts are shared, and “inter”-interbook surfing is possible. However, in InterBook, each interbook can be created by a different author, at different times. The AHA! solution would require a single author to submit everything at a single time. In the future, AHA! could be extended by ontology-based concepts, instead of application-based.

6.2 Improving Usability

As we indicated in Section 5.1, we ran into the limitation of the Graph Author, when we tried to display large numbers of concepts and relations. Because of the large number, the overall picture was lost. We would like to propose two areas of future work to overcome this problem.

First, the ability to hide groups of concepts (e.g. of a certain type, of manual selection, or everything not connected to the currently selected concept), plus their relations, would allow to dramatically reduce the number of displayed objects, and thus increase the cognitive value of the remaining objects. Another (more limited) option would be to separate concepts by different color or shape.

Second, currently the concepts are manually organized. This becomes a very tedious and difficult process when a large number of concepts is involved. Currently a lot of research is being done in the area of displaying large graphs (concept-trees, ontology, etc). Using such an automatically organized graph for the display of the concept structure would most likely increase its usability for large numbers of concepts and relations.

In Section 4.3.4 we mentioned the limitation on annotating internal links, because AHA! can not automatically create a link to a resource listed as default fragment for a concept. This could possibly be solved if the default fragment would be listed as normal concept resource, instead of separately as default fragment. However, the feasibility of this is currently unknown to us.

6.3 Improving InterBook/latA

KBS Hyperbook (Section 1.3) sorts and annotates links in its Glossary section based on the type of link. This increases the understanding of the users concerning which link to choose. With the InterBook annotation it is very easy to add this information (compare our extension with *introduces*, *exemplifies*, *is-a* and *part-of*). However, the relation *type* is lost when an application is imported into AHA!; only the *source and destination*, and *functionality* is stored for each concept relation. If, instead, the relation type would be stored with the source and destination, a duplication of information could be avoided, while the relation type is maintained for possible adaptation. A different solution for manually created relations (through the Concept Editor, in which low-level condition-action rules can be created) might have to be created.

Currently still a very young research area is the automatic indexing of documents (through data-mining). We foresee in the future a connection between our (or similar) work and such work, removing even more effort required from the author, when converting a textbook to an adaptive ET. Where the automatic indexing might be able to obtain the concepts introduced in a section (and possibly through e.g. an ontology also the prerequisite concepts), our translation process can take these extracted concepts, and generate the adaptive application from them, similarly as it now uses the manually added annotations. By further reducing the required extra effort, it might become increasingly easy (and custom) to create adaptive ET.

Conclusion

Not all developed AHS are currently still being further developed, or maintained, even though they might have features that more current systems are missing. To address this issue we intended to create a reusable translation to conserve classic systems' content, and their (unique) features. As target system we choose AHA!. It is an general-purpose system that supports both a new level of concept structure translation, and a Layout Model for easy recreation of the Look and Feel. As source system we choose InterBook, both because of its extensive content, and the special feature of easy content creation through RTF files, a feature for which AHA! has no counterpart.

We created a reusable concept structure level translation and recreated the Look and Feel of InterBook on AHA!. Using these, we made it possible to easily translate existing (original InterBook) courses to AHA! applications, as well as the (easy) development of new AHA! applications through our authoring-through-translation approach on the level of concept structures; currently a C programming application is under development using this approach (see also Appendix B for more details around that project).

Thus, we have shown with our translation that it is possible to create a translation process with classic InterBook ET as source, as to automatically generate an application on a AHA!, which has both a similar functionality and look and feel as the original course. This kind of translation is interesting because it allows preserving the effort and knowledge incorporated in such courses.

By creating a reusable translation from (classic) InterBook, with good support for content creation, to AHA! (with limited support in this area), we have shown that it is possible to use (extended) functionality from one system, and then, through translation, use this functionality to enhance other systems (lacking support in that area).

Although our translation only involves a single AHS as source, we see the generality of our approach, because, although some programming (in JAVA) can be required for the creation of new views, no change of the core code on the AHA! engine is required for new translations, and new views can be created without involving (or changing) any original views. Therefore it is theoretically possible to make unlimited additional translations, with each using their own views, while they will neither burden the AHA! core, nor influence the working of other (previously translated) systems on AHA!. Also, our translation (of concept structures and content) is independent of AHA!; using a different system as source, with a different translation (using the same approach) makes it possible to create the same kind of input files for AHA!, while not influencing our current translation at all.

The limitation of our conclusion is, however, coming from the fact that we only attempted to translate a single system. Although it is an important step towards the final goal of conservation and preservation of classic AHS and their content, and towards the separation of the development of *adaptive hypermedia applications* from the development of *adaptive hypermedia systems*, more research and translations are required to prove the generality of this approach.

References

1. Benjamin S. Bloom. *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*. Longman Publishing Group, 1956.
2. Paul De Bra, Licia Calvi. *Creating adaptive Hyperdocuments for and on the Web*. Proc. WebNet'97, pp. 149-155, 1997.
3. Paul De Bra, Licia Calvi. *2L670: A Flexible Adaptive Hypertext Courseware System*. Proc. 9th ACM Conference on Hypertext, pp. 283-284, 1998.
4. Paul De Bra, Geert-Jan Houben, Hongjing Wu. *AHAM: A Dexter-based Reference Model for Adaptive Hypermedia*. Proc. 10th ACM Conference on Hypertext, pp. 147-156, 1999.
5. Paul De Bra, Ad Aerts, Brendan Rousseau. *Concept Relationship Types for AHA! 2.0*. Proc. E-learn 2002 Conference. pp. 1386-1389, 2002.
6. Paul De Bra, Tomislav Santic, Peter Brusilovsky. *AHA! meets Interbook, and more...* Proc. E-Learn 2003 Conference. pp. 57-64, 2003.
7. Paul De Bra, Natalia Stash, David Smits. *Creating Adaptive Textbooks with AHA!*. Proc. E-Learn 2004 Conference, pp.. 2588-2593, 2004.
8. Timothy Brailsford, Helen Ashman, Craig Stewart, Mohamed Ramzy Zakaria, Adam Moore. *User Control of Adaptation in an Automated Web-Based Learning Environment*. ICITA 2002 – 1st Conference on Information Technology & Applications, 2002.
9. Peter Brusilovsky. *Methods and techniques of adaptive hypermedia*. User Modeling and User-Adapted Interaction v.6, pp. 87-129, 1996.
10. Peter Brusilovsky, Elmar Schwarz, Gerhard Weber. *A Tool for Developing Adaptive Electronic Textbooks on WWW*. Proc. WebNet'96 Conference, pp64-69, 1996.
11. Peter Brusilovsky, Steven Ritter, Elmar Schwarz. *Distributed intelligent tutoring on the Web*. Proc. 8th Conference on Artificial Intelligence in Education, pp. 482-489, 1997
12. Peter Brusilovsky, J. Eklund, Gerhard Weber. *Web-based Education for All: A Tool for Development Adaptive Courseware*. Proc. 7th World Wide Web Conference, pp. 291-300, 1998.
13. Peter Brusilovsky, Tomislav Santic, Paul De Bra. *A Flexible Layout Model for a Web-Based Adaptive Hypermedia Architecture*. Proc. AH2003 Workshop, pp. 77-86, 2003.
14. Peter Brusilovsky. *KnowledgeTree: a Distributed Architecture for Adaptive E-Learning*. Proc. 13th World Wide Web Conference, pp. 104-113, 2004.
15. Alexandra Cristea, Davy Floes, Natalia Stach, Paul De Bra. *MOT meets AHA!*. Proc. PEG'03 Conference, 2003.
16. Peter Fröhlich, Wolfgang Nejdl, Martin Wolpers. *KBS-HYPERBOOK: An Open Hyperbook System for Education*. Proc. ED-MEDIA Conference on Educational Multimedia and Hypermedia, 1998.
17. Frank Halasz, Mayer Schwartz. *The Dexter Hypertext Reference Model*. Communications of the ACM, vol 37, nr 2. pp. 30-39, 1994.
18. Nicola Henze, Wolfgang Nejdl. *Adaptivity in the KBS Hyperbook System*. Proc. 2nd Workshop on Adaptive Systems and User Modeling on the WWW, 1999.

19. Adam Moore, Craig Stewart, Duncan Martin, Timothy Brailsford, Helen Ashman. *Links for Learning: Linking for an Adaptive Learning Environment*. Proc. IASTED Conference on Web Based Education, 2004.
20. Tom Murray. *MetaLinks: Authoring and Affordances for Conceptual and Narrative Flow in Adaptive Hyperbooks*. International Journal of Artificial Intelligence in Education, vol. 13, 2002.
21. Wolfgang Nejdl, Martin Wolpers. *KBS Hyperbook – A Data-Driven Information system on the Web*. Proc. 8th World Wide Web Conference, 1999.
22. Ewald Ramp, Paul De Bra, Peter Brusilovsky. *Authoring and Delivery of Adaptive Electronic Textbooks made Easy*. Proc. E-Learn 2005 Conference. To be published, 2005

Appendix A. R2Net

The commercially available R2Net (from Logictran) is a program designed to translate RTF files into HTML or XML files. It evolved from RTF2HTML used by InterBook, and is highly scriptable. We used this feature to create the required AHA! files. We used its TOC creation to generate a page concept for each section. As concept name we use the R2Net generated page name, the combination of the filename with part of the header name. This causes page concepts to automatically obtain unique names, plus that they will always be highly distinctive to which application they belong.

Global concepts are created for each section after the detection of a “header 1”-formatted “Glossary” string. Unlike contents files for page concepts, which are linked as concept resource, we link the content files for the global concepts as default fragment. This allows the reusable content described in Section 4.3.3. Concepts that are encountered only at the start of sections (within relational annotations) are also created as global concepts, and (besides the respective relation information with the currently processed section) saved without resource or fragment. All global concepts are saved as children of an abstract concept with the name “glossary_ontology_AHASystemDoNotUse” to avoid duplication, and with title “Glossary”. A lookup list is updated and inspected to avoid duplicate global concepts through this approach. Global concepts get the (restricted) version of the used concept name in the RTF file. Certain limitations are placed upon the AHA! concept names, for example, a dot (.) exclamation mark (!), or space are not allowed within concept names.

Concepts declared inline are saved as fragment concepts, because they are not part of the ontology, and therefore should not show up in the Glossary Window. However, they are stored as children of the abstract glossary_ontology_AHASystemDoNotUse concept too. For concept name, the (restricted) inline declared concept name is used.

To obtain the InterBook-specific colored bar above the text, we include the following fragment above the text of each section:

```
<if expr="<RTF_File_Name>.<R2Net_page_name>.suitability">
  <block>
    <hr color="green" size="5" width="50%" />
  </block>
  <block>
    <hr color="red" size="5" width="50%" />
  </block>
</if>
```

Where <RTF_File_Name> is the name of the RTF input file (and the application name), and <R2Net_page_name> is the page concept name, as generated by R2Net.

The [Back] and [Continue] “buttons” are also generated by R2Net. Preceding the above fragment we include a *conditional* link to the *page of the previous section*, with hot word “[Back]”, which is translated by AHA! to an annotated link to the appropriate concept. The [Continue] link is treated in a similar fashion, linking to the next page, below the section contents. No header or footer files are currently included.

Appendix B. Talking Bloom: AHA! Communicates with Sedona

Introduction

Many researchers have attempted to let several AHS communicate in a sensible fashion. [11] Describes a 1 to 1 approach, [14] uses a master-slave system where everyone reports to a single UM server, and [4] suggests a system-independent communication interface, building upon the AHAM-definition of user model. While the master-slave approach with a single UM server assumes that “the UM server knows best” and that all application servers will rely upon this to obtain the correct values for a User Model (for a specific user), AHAM focuses on completely separate systems, each with their own UM. The main problem the authors foresee in that case is the interpretation of a concept C from another system in its own concepts, and the appropriate conversion from knowledge-values.

In a new research project at the University of Pittsburgh, several researchers are attempting to create a new architecture based around a central *Ontology Server* (-network). This Ontology Server (OS) will be able to translate between different ontologies, and duplicate/maintain a small part of the User Models from all registered AHS (only the knowledge values are duplicated). Because the OS is able to map between different ontology (in which the User Models are expressed), it is the most appropriate place to interpret concepts between different ontologies. The second problem foreseen by the AHAM authors, appropriate conversion of knowledge-values, is handled in a slightly different way. When we compare different (educational) systems, we can notice that they often work on different levels when expressed in Bloom’s Taxonomy [1]. Therefore, expressing the knowledge stored on the OS in probabilities of mastering the respective Bloom Level already partially overcomes the problem. Instead of having to communicate in any knowledge-value type, an AHS only needs to be able to translate between its own values and Bloom levels (between 0 and 1). Of course, it still need to trust the other systems to report (and translate) correctly, but we want to restrict ourselves here to mentioning the possibility to use different calculations from Bloom level probability to internal representation, based upon the reporting system.

To create a test system, it was decided to use KnowledgeTree (with Cumulate UM-server) and AHA!. An Ontology Server (Sedona) is being created, through which the mentioned systems will communicate. An extension to AHA! was required to allow for this communication which we will discuss next. The extension was kept to a minimum because of certain restrictions, and only the push-interface (toward Sedona) is minimally implemented for communication. For this test, an AHA! application is being developed on C programming using our newly developed authoring-through-translation approach, for a C programming class thought there using KnowledgeTree. Once this common ground is created, the goal is to research the possibility to allow the communication of user knowledge obtained through AHA! and the KnowledgeTree activity servers. Each of the systems for this new architecture will have its own specialization and is able to teach the user on different levels according to Blooms Taxonomy. Exchanging the knowledge about the different levels between the systems will hopefully allow for improved (initialization of and) adaptation based upon the user models.

Reporting to the Ontology Server

If we want to allow the OS to report knowledge, we need to first provide it with knowledge. For this, we need to report to the OS in a special format for every knowledge mutation on the (local) AHS. In a new (globally accessible) class within AHA!, ShareProfile, we have defined the necessary functionality, which is called upon, for each updated knowledge attribute, from

the function `ProfileUpdate.processAttributeUpdate()`. This guarantees that only newly updated knowledge attributes are reported. In the called `ShareProfile.report()` we then filter out everything except the global concepts, (which, for IatA applications, span up the ontology) and use the integer value to linearly define the probability for knowledge about the first Bloom Level. (A value of 100 or larger defines a probability of 1).

Updating AHA! through requests

Because we also would like to incorporate knowledge reported by other systems to the OS within AHA!, we extended the `ShareProfile` class with a request function. We call this function once each time a user logs-in (or registers) to an application that is registered with an OS.

The returned knowledge is subsequently filtered and converted. We currently filter out and discard knowledge from the same Bloom Level as AHA! produces (recal), to simplify things to ourselves and avoid the need to combine that knowledge and ours. (The probability of Knowledge on higher levels is used if the probability is higher than 50%. We do not distinguish between systems. If we receive a high enough probability, we store that in the AHA! database as 150. This is distinguishable from knowledge obtained through AHA! itself, which has a maximum value of 100. We use this to assign a 3rd level of checkmark behind a concept (in an IatA application) to indicate this higher level of Knowledge, obtained externally (similar to the used convention in InterBook).

Future Work

In [11] the actual exchange of knowledge between InterBook and PAT online is described. The basic architecture for a similar interaction is being created at the University of Pittsburgh. A preliminary test is being set up using an extended version of AHA! and KnowledgeTree. Although no real conclusions can be made yet, besides that it seems to be possible to extend an AHS like AHA! with the necessary interface, a lot of future work with respect to AHA! can be defined when this test is successful. Preliminary tests have shown that AHA! from its part does report nicely all new knowledge without overhead, and although limited, it is able to incorporate the external knowledge within its applications, while not interfering with other applications (not incorporated in the test).

If the main test is also successful, more elaborate and integrated support from the AHA! system itself for this exchange of knowledge might be needed (beyond the current limited extension), and concept relations might have to be tuned to incorporate the additional information. Another possible further extension to AHA! with respect to this knowledge exchange might be the extension of the graph authoring process by additional kinds/levels of relations to map internal knowledge to Blooms levels for each concept type, and vice versa based upon the reporting (external) system.