

AHA! meets Auld Linky : Integrating Designed and Free-form Hypertext Systems

David Millard, Hugh Davis, Mark Weal
University of Southampton
Southampton, UK
{dem,hcd,mjw}@ecs.soton.ac.uk

Koen Aben, Paul De Bra
Eindhoven University of Technology,
Eindhoven, The Netherlands
{s458238,debra}@win.tue.nl

ABSTRACT

In this paper we present our efforts to integrate two adaptive hypermedia systems that take very different approaches. The Adaptive Hypermedia Architecture (AHA!) aims to establish a consistently organized, strictly designed form of hypertext while Auld Linky takes an open and potentially sculptural approach, producing more freeform, less deterministic hypertexts.

We describe the difficulties in reconciling the two approaches. This leads us to draw a number of conclusions about the benefits and disadvantages of both and the concessions that are required to combine them successfully.

Categories and Subject Descriptors

H.5.4 [Hypertext/Hypermedia]: Theory; H.1.1 [Systems and Information Theory]: General systems theory

1. INTRODUCTION

Within the Hypermedia research community there have been two implicit approaches to the design (and authoring) of Hypertexts. Some systems, such as Hyper-G [3] and Intermedia [17], took designed approaches which are *deterministic* in the sense that an author always understands the navigational paths available to their reader at any one time. Even when the systems are adaptive, there is tight control over which links are available and the possibility space is small.

Other systems, such as Microcosm [15] and Chimera [2], took a more *freeform* approach. Because the hyperstructure was assembled at run-time from many potential sources an author knew much less about the links available to a reader, as such we can describe these systems as non-deterministic.

Broadly speaking these two approaches characterise the two trends of Hypertext research, known as Adaptive Hypermedia (AH) and Open Hypermedia (OH), that have evolved in parallel since the late 1980's.

AH researchers have been concerned with applying the methods found in Artificial Intelligence, Intelligent Tutoring Systems and User Modelling Communities to Hypertext, in order to create

dynamic systems that adapt the hyperstructure at run-time to the user's current needs [10, 8].

OH researchers have concentrated on investigating and separating the structural layers of a hypertext system. OH Systems store link structures and content separately and combine the two together at run-time, allowing the structures to be processed separately and easily applied to a wide variety of media [16, 15].

Researchers from the Eindhoven University of Technology have recently been working with the IAM Group at the University of Southampton with the aim of better understanding the relationship between Eindhoven's AH system, called Adaptive Hypermedia Architecture (AHA!) [8], and Southampton's most recent OH system, Auld Linky [18].

Our work is motivated by a desire to apply the AH structures described in previous work [5] with an existing practical adaptive system. The objective is to discover how the structural approach affects the processes of adaptation within AHA and to see if the OH paradigm can support evolving structures alongside a more pre-determined adaptive model.

In this paper we describe how we have attempted to *combine* AHA! with Auld Linky using ideas from Sculptural Hypertext [6, 22].

Auld Linky is an Open Hypermedia System (OHS) that might be described as adaptive in that it selects the hyperstructure at run-time according to context. However, it is not necessarily driven by the sophisticated reasoning that characterises an Adaptive Hypermedia System (AHS) like AHA!.

Because of this difference we have been forced to reconcile the two Hypermedia approaches in our integration work, balancing the different methodologies of AHS and OHS in an attempt to understand each in the terms of the other.

Section 2 describes AHA! and the scripting language that has recently been introduced to it.

Section 3 describes Auld Linky and the Sculptural Hypertext authoring paradigm.

Sections 4 and 5 present our design for how the AHA! scripting language could be used to produce sculptural link bases that can be manipulated by Auld Linky and also how they can be used to augment a running AHA! application. We also describe some of the problems we have had to confront as a result of the tension between the different methodologies.

In Section 6 we reflect on the new understanding of how the two systems and their methodologies relate to one another, and propose a future integration strategy for methodologically different systems, based on this understanding.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HT'03, August 26–30, 2003, Nottingham, United Kingdom.
Copyright 2003 ACM 1-58113-704-4/03/0007 ...\$5.00.

2. ADAPTIVE HYPERMEDIA ARCHITECTURE (AHA!)

The Adaptive Hypermedia Architecture (AHA!) is a generic system that provides authors with a web-based runtime layer and design tools for the creation and implementation of adaptive hypermedia applications.

We believe that AHA! is the most suitable hypermedia architecture to use for our integration work, because it deals with adaptive hypermedia from a detailed conceptual point of view and therefore has the potential to create complex interacting concept structures which are very deterministic.

Other architectures, like KBS Hyperbook [20] or Interbook [1], emphasize high-level adaptivity which means that it is easier to add new structures to an existing application. However, adaptive hypermedia often requires more complex components with detailed dependencies. AHA! allows us to look at how these very deterministic designs contrast with the more free-form OHS approach.

2.1 Description of AHA!

To construct an application with AHA! an author needs to create: *page templates* and *concepts*.

A Page template is an XHTML file that contains conditionally included fragments. For each fragment there is a Boolean expression using concepts and attributes from the user model. This expression is evaluated at run-time to decide whether the fragment will be shown to the user. Conditional fragments can be used for many purposes, including conditional pieces of information (e.g. explanations) or for opening and closing menus and submenus when navigating through a "sitemap".

Concepts are named entities that have attributes and condition-action rules for updating attribute values in the user model. The rules are triggered by a page access and interact with each other to calculate a "next" stable state space for the user.

The AHA! runtime component is an adaptive engine, consisting of a set of Java servlets, which maintains a user profile and which uses that profile and condition-action rules to render the appropriate contents into a HTML file for display. (This rendering includes selecting colours for links and the conditional inclusion of fragments.)

2.2 The AHA! Approach

AHA! allows for very flexible cooperation between concepts. (Every concept is allowed to change the state of another concept or page template and thus trigger a variety of further possible changes.) However, authoring using these condition-action rules directly is difficult and can easily lead to rule sets that produce unpredictable results, or that enter an infinite loop when executed. Therefore an authoring tool was developed that lets authors define an application using a *concept hierarchy* and *concept relationships* of some predefined types [9]. This *graph author* tool (introduced in AHA! version 2.0) helps non-expert authors to make use of AHA!'s adaptive features in a controlled, structured way. However, it does not give authors access to the full functionality of AHA!.

2.3 An AHA! Scripting Language

The limited functionality of the graphical authoring tools was not the only trigger for a new authoring approach. To make use of the full capacity of AHA, one has to understand the architecture down to a very detailed low level. AHA! offers an authoring tool that hides the XML syntax from the author, but the meaning of AHA!'s condition-action rules and how the adaptive engine executes them must be understood in order to use the full functionality of AHA!.

An experimental scripting language was designed to allow com-

```
import ahs.Template.*;
import ahs.Concept.*;
import ahs.Link.*;

Concept SimpleAHS {
  Link progressLink =
    new Link(SimplePage, "press link for progress");
  String Time = new String("Day");
  Template SimplePage;

  Concept Main {
    if ( SimplePage.access ) {
      case ( Time ) {
        ( "Day" ) { Time = "Night"; }
        ( "Night" ) { Time = "Day"; }
      };
    }
  }

  Template SimplePage {
    if ( Time = "Night" ) {
      println("Good Night.");
    };
    if ( Time = "Day" ) {
      println("Good Day.");
    };
    progressLink;
  }
}
```

Figure 1: A Simple Adaptive Page in the experimental AHA! Scripting Language

plex interactions to be encoded easily. AHA! contains a flexible adaptive engine for a concept and rule structure in which any concept can be programmed to have a sophisticated influence on any set of concepts and vice versa. Establishing *direct* relationships between concepts is relatively easy. However when complexity of the relations between concepts (and the rules to implement them) increases, the *indirect* dependencies of templates and other concepts become very complex to author.

The new scripting language tries to simulate an object oriented approach for the construction of the AHS. By forcing an author to program in an object oriented style we gain more manageable concepts, because relationships and meaning are carefully structured into a programmable element. The AHS is therefore divided into a hierarchical tree of elements. A tree where each node is a subsystem, equipped with a program to control its set of subtrees. The leaves of a tree are used to store user data, such as the current time or a temporal interaction with the user (a link click or page access).

Figure 1 shows an example of the new scripting language. For reasons of brevity it is very simple (although more complex scripts would be similarly structured): the greeting displayed in the page template changes each time a user clicks on an "increase time" link.

Because the scripting language expresses user navigation and page construction, it is in fact independent of the AHA! system. Since the scripting language is a summary of the use and functionality of AHA!, it was soon realised that it might form a basis for some collaboration between AHA! and other systems. Scripts can be compiled into other hypermedia formats, provided that these are sufficiently powerful. In particular we looked at using Auld Linky and producing a Sculptural Hypertext from the original AHA! scripts.

3. LINKY AND THE SCULPTURAL HYPERTEXT PARADIGM

Auld Linky is a structure server that is specifically designed to be used as an OHS with contextual support. It is a stand alone process that manages an XML “linkbase” of association structures expressed in FOHM [19] and provides pattern matching services via HTTP, modifying the structures served according to the declared context of the querying client.

It is important to note that Auld Linky is not just a server of hyperlinks. Auld Linky is part of a new trend in OHS research to create structural systems that are capable of handling a wide range of relational structures (in addition to links) such as tours, trails and virtual documents. A major advantage of modelling hypertext in this way is that the resulting structures are consistent in that they may all refer to one another and are all subject to the same processes.

While larger systems allow for an open set of operations on their structure, Linky is specialised to deal with a particular type of structure filtering based on a declared context. This allows for more sophisticated views on the structure than just mixing and matching linkbases. Contexts in Linky are thus more akin to profiles in AHS research, or link processes in the HB1/SB1 series of systems [21], than they are to contexts (or linkbases) in other OHS work.

Users querying Linky do so within a context (defined with open-ended metadata) and Linky provides a view of *all* the loaded linkbases given that context, allowing for unions and intersections of structure that cannot be handled with multiple linkbases.

Recently we have shown how Auld Linky’s form of contextual Open Hypermedia can be used to implement many of the techniques common in Adaptive Hypermedia Systems [5]. In fact Auld Linky can be seen as an Adaptive Engine for generic hypermedia structures, adapting both navigational links and content. We have also been exploring the other types of hypertext that become possible.

At the 2001 ACM Hypertext conference we presented work on a contextual version of the generic link [22]. At the same event Mark Bernstein presented a collection of what he described as “strange hypertexts” which were authored and/or experienced in different ways to traditional node/link hypertexts [6]. He coined the term *Sculptural Hypertext* to describe a hypertext authoring paradigm where all the nodes are initially interconnected and where the author uses pre-conditions and actions to remove links at runtime to produce the final experience.

Auld Linky linkbases of contextual generic links can also be described as sculptural hypertexts. Figure 2 shows a link from a sculptural linkbase. The source of the link is variable (will match everything) which means that this is a link that may be viewed from any source document. It is the context (the conditions), and not the location in the writing space, that determines which links the reader can see. The Behaviour objects store the appropriate actions.

Open Hypermedia can be thought of as non-deterministic because some of the hyper-structure relies on context and structure that is not known until runtime. Sculptural hypertext is an extreme form of this, as *all* the connections are dependent on context and none of the final structure emerges until runtime.

We refer to sculptural hypertexts as freeform as they encourage an author to concentrate on the individual text fragments and their associated conditions and actions (the context in which they can be seen and how they effect that context when read).

Sculptural hypertext makes no apologies for the effects of this approach, rather it is a design methodology that encourages the author to break free of any high level design and focus only on

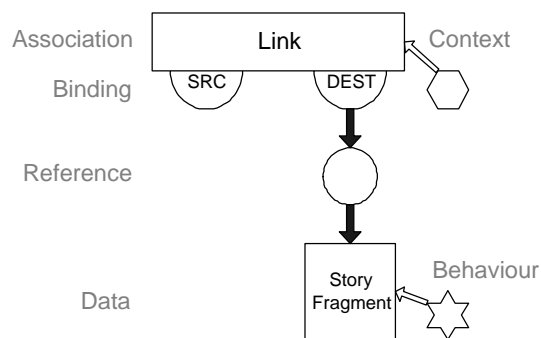


Figure 2: A Contextual Generic Link

isolated glimpses of a writing space.

4. SCRIPTING SCULPTURAL LINKBASES

Given that AHA! is in practice a state machine, it should be possible to capture the states and transitions of a given application using links, conditions and actions. These elements could be used to produce a sculptural linkbase to be fed into Auld Linky.

The sculptural links produced could be explored separately from AHA! using a sculptural links renderer or fed back into AHA! at runtime to be presented in addition to the links that AHA! would normally provide. This is advantageous as the freeform nature of the sculptural links would mean that links between otherwise separate AHA! applications would automatically emerge even though they had not been explicitly designed by the authors of those applications.

Gathering the set of conditions and calculating appropriate actions for each state in an AHA! application is non-trivial. The AHA! engine reasons about the user and the content to deliver. It contains sequences of assignments and stores the possible transitions to new assignments and sequences. Dependencies within the script need to be analysed by the compiler in order to generate the conditions for the sculptural context objects.

Even if we are not able to glean all the implicit conditions contained in the AHA! script we can still produce a useable sculptural linkbase. However, the sculptural version of the application would be “looser” than its AHA! counterpart, with more links available to a reader at any one time.

The two main components of an AHA! application that must be translated into the linkbase are the Page Templates and Concepts.

4.1 Page Templates

The first “problem” with pages in AHA! (ignoring the template aspect with conditional fragments for a moment) is that pages are treated as concepts by the AHA! system but perhaps not considered as concepts by an author. The consequence is that an author may define rules in terms of pages instead of “higher level” concepts. An author may express that *Page3* should become available only after the user has visited *Page1* and *Page 2*. While this can be directly encoded into sculptural links it results in context objects that only have meaning in a very restricted scope. What we really need is for *Page3* to say that it is only available once two particular *subjects* have been introduced to the user, and for *Page1* and *Page2* to declare that they introduce those subjects. (This can be done using AHA! rules, but AHA! does not enforce doing it this way.)

The problem can be summarised by saying that in AHA! sequences are explicit but their semantics may not be. In other words

the author explicitly defines routes through the writing space but is not required to define why those routes exist (why one piece of content sensibly follows another).

In contrast, in Sculptural Hypertext it is the sequence that can be implicit and the semantics that must be explicit. The order in which content may be viewed emerges from the interaction of the conditions and actions.

To successfully derive a sculptural linkbase from an AHA! script the explicit semantics need to be included. This would require some changes to the AHA! scripting language in order to encourage authors to declare the semantics of their pages in respect to a particular vocabulary or ontology. Once this is done all sculptural links that share that vocabulary, whatever their application of origin, may be used together.

In AHA! the “template” part of pages are structures that represent a user’s view on a page. This view, and the fragments that are included, change according to the state of the system. Although our previous work with Sculptural links has used static “fragments” of text as the destination of sculptural links, it is possible to use other FOHM structures instead. The one that we have used in other applications and which is the closest equivalent to an AHA! page template is the *virtual document*.

The members of the virtual document are designed to be conditionally rendered into a single document for display. Virtual documents can still be viewed as a single document, the appearance alters according to the viewer’s context. Templates, while functionally equivalent, are semantically different as they often contain mutually exclusive members. They represent a framework within which several different documents may be rendered.

4.2 Concepts and Rules

In AHA! concepts can be related such that changes in one Concept effect the status of others. For example, the concept of *cooking expert* might be modified as the result of a user triggering several lower concepts, such as *Thai cooking expert*, *Italian cooking expert*, etc. These in turn might be triggered as a result of a user activating even lower level concepts, for example once they become familiar with a certain number of recipes.

In Linky context objects are just collections of attribute value pairs and cannot refer to each other in any way. In order to represent these interconnected concepts they must be “flattened”. So in our *cooking expert* example, every time a page relies on a viewer being a cooking expert, the context in linky must exhaustively list all the recipes from all the styles that they must know.

We have previously acknowledged that Linky’s representation of context (the format of the actual context objects) is a basic one and have discussed using some form of ontological representation to enhance it [18].

Allowing context objects to refer to one another would be a first step towards this and would allow Linky’s contexts to more accurately reflect the AHA! Concept hierarchy. It also requires context objects to become first class and therefore reusable (currently in Linky they are anonymous). Experience with AHA! would indicate that this is necessary for any level of complexity in the design of contexts.

4.3 A Simple Example

In this section we show how the simple example of Figure 1 may be translated to AHA! Concepts and Page templates, and how it may be translated to Auld Linky structures.

Figure 3 shows the result of compiling the script into AHA! concepts and an associated page template. The result is actually simplified to make it easier to understand. Not only are the assign-

ments shown in a more compact syntax, the rules are also made non-propagating to avoid an infinite loop. (In a future version of AHA! there will be a “case” statement, but for now a correct compilation would require an additional attribute to avoid an infinite loop while keeping the actions propagating.)

The figure shows that in the translation, code has to be moved. Any piece of program code that depends on `SimplePage.access` must be moved to actions associated with the `access` attribute of `SimplePage`. This is a nice illustration of the fundamental difference between AHA!’s way of associating actions (that define user model updates) with the events or with attribute updates, making the “flow” of updates difficult to trace, whereas in the scripting language the actions are written in a program that clearly shows the order in which attributes of concepts are updated.

```
...
<concept>
  <name>SimpleAHS</name>
  <attribute name="Time" type="string" isSystem="false">
    <default>Day</default>
  </attribute>
</concept>

<concept>
  <name>SimplePage</name>
  <resource>SimplePage.xhtml</resource>
  <attribute name="access" type="bool" isSystem="true">
    <generateListItem isPropagating="false">
      <requirement>SimpleAHS.Time="Day"</requirement>
      <>trueActions>
        <action>SimpleAHS.Time:="Night"</action>
      </trueActions>
    </generateListItem>
    <generateListItem isPropagating="false">
      <requirement>SimpleAHS.Time="Night"</requirement>
      <>trueActions>
        <action>SimpleAHS.Time:="Day"</action>
      </trueActions>
    </generateListItem>
  </attribute>
</concept>
...
<xhtml>
  <header>
    <title> Example of a page template </title>
  </header>
  <body>
    <if expr="SimpleAHS.Time==Night">
      <block> Good Night. </block>
    </if>
    <if expr="SimpleAHS.Time==Day">
      <block> Good day. </block>
    </if>
    <a href="SimplePage" type = "conditional">
      increase time
    </a>
  </body>
</xhtml>
```

Figure 3: The Concepts and Page Template generated by the script in Figure 1

Figure 4 shows the same information encoded as a Linky virtual document. In this case the bindings have context attached to them which determines the membership of the document and the data objects have the behaviour that changes the user context.

```

<association id="vdoc001">
  <relationtype>virtualdoc</relationtype>
  <structure>list</structure>
  <feature>position</feature>
  <binding>
    <featurevalue feature="position">1</featurevalue>
    <reference>
      <data><url><![CDATA[Good-day.xml]]></url>
      <behaviour><event>ondisplay</event>
      <behaviourvalue key="time">night</behaviourvalue>
    </reference>
    </data>
  </binding>
  <binding>
    <featurevalue feature="position">2</featurevalue>
    <reference>
      <data>
        <datacontent><![CDATA[a good night]]></datacontent>
        <behaviour><event>ondisplay</event>
        <behaviourvalue key="time">day</behaviourvalue>
      </data>
    </reference>
    <context>
      <contextvalue key="time">day</contextvalue>
    </context>
  </binding>
  <binding>
    <featurevalue feature="position">2</featurevalue>
    <reference>
      <data>
        <datacontent><![CDATA[a good night]]></datacontent>
        <behaviour><event>ondisplay</event>
        <behaviourvalue key="time">day</behaviourvalue>
      </data>
    </reference>
    <context>
      <contextvalue key="time">night</contextvalue>
    </context>
  </binding>
  <behaviour>
    <event>onAHADisplay</event>
    <behaviourvalue key="template">template1</behaviourvalue>
  </behaviour>
</association>

```

Figure 4: The Concept and Page Template from Figure 3 expressed as a Linky Virtual Document

5. FREEFORM LINKS BETWEEN DESIGNED APPLICATIONS

In AHA! a closed adaptive engine has to perform all the data transformations and as a result a freeform collaboration between separate adaptive applications is currently not possible (although links towards external web sites and embedded external texts are supported).

It is possible to design constrained collaboration between applications by allowing high level concepts to manage the interactions between their state spaces. These interactions are deterministic and must be described at compile time. They conform to AHA!'s tight design principles but do not allow for serendipitous linking (with automatic links between applications being generated at run-time).

By comparison if we compile two AHA! applications into sculptural linkbases and load them into a single instance of Linky we find that, as long as the applications share the same concept vocabulary when describing state, the two linkbases automatically merge into one large application. I.e. when in the middle of one application we see not only the links to other pages in the same application but also links to appropriate pages in other applications.

Our intention was to give AHA! access to this form of opportunistic linking by expanding the architecture of AHA! with an additional linkbase of sculptural links that would be added at runtime.

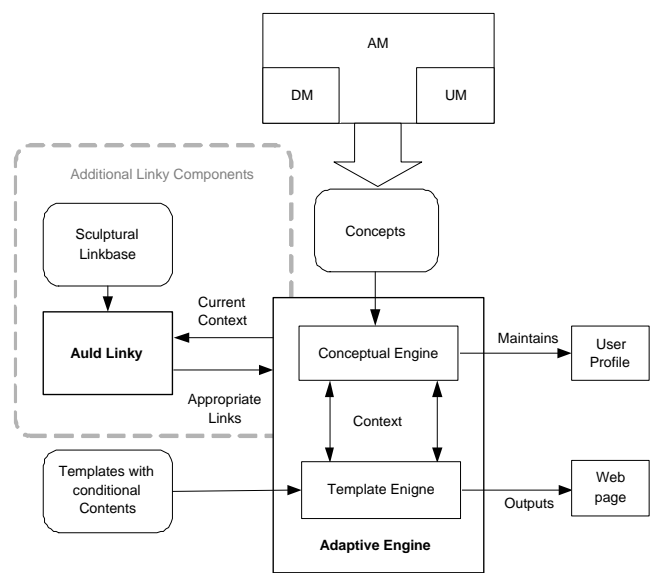


Figure 5: Linky augmenting a running AHA! System

Figure 5 shows how Linky could be used to augment a running AHA! application. The sculptural linkbase generated from the script is loaded into Auld Linky. The AHA! adaptive engine maintains the current user model (based on the AHA! concepts). When a page is displayed it translates the current state (user model) into a Linky context object and uses this as the context for a query to Linky to retrieve any applicable links.

Linky responds with a set of available links and the adaptive engine then uses the Template Engine to render them on the final page as navigation choices. If one of the links is followed the Conceptual Engine interprets the Behaviour attached to the links. This can cause difficulties when the link is between two different AHA! applications. So care must be taken to ensure that AHA! is initialised properly for the new application.

5.1 Initialising AHA!

Sculptural links allow a user to “drop-in” to the middle of an AHA! application from an external source (e.g. a page in another application). This inter-application linking causes two problems:

- *How to Render a Page Template.* AHA! sometimes stores several fragments as mutually exclusive views on a single page template. Since the sculptural links refer to the template there is no natural way for AHA! to calculate which view is appropriate to display on arrival.
- *Maintaining Consistent State* AHA! depends on a consistent sequence of states. A user who appears suddenly in the middle of an application will not have visited the expected previous states and therefore will not have initialised that application appropriately.

5.1.1 Rendering Page Templates

In the sculptural linkbase each template is represented by a virtual document, made up of all the possible members of the template. The AHA! scripted conditions are transformed into context objects attached to each member. In this way, in different contexts, the virtual document has different members.

For each *view* of a template in AHA! (i.e. combination of valid members) a sculptural link is authored in the linkbase. The links

can only be seen in the correct context and when they are followed the virtual document is retrieved in the same context and thus the appropriate view is shown. Unfortunately this is only of use to the Sculptural Servlet; AHA! still needs to know which page template to use and which view to render.

To cope with this the script compiler needs to add behaviour objects to the links and virtual documents in the linkbase. Each virtual document is tagged with the name of the template in AHA! it describes (for example, the virtual document in Figure 4 represents (“Template1”) and each link is tagged with a sequence of true/false flags describing how AHA! should display it.

When a sculptural link is followed in AHA! the system ignores the virtual document object and uses these two behaviours to locate and display the appropriate template.

Figure 6 shows the two sculptural links that would be authored to the virtual document in Figure 4 (one for each valid view). The context on the link ensures that they can only be seen in the appropriate context (i.e. the link for a Morning greeting is only shown when the user’s context is Morning).

```
<association id="link001">
  <structure>link</structure><feature>direction</feature>
  <description>Morning</description>
  <binding missing="variable">
    <featurevalue feature="direction">src</featurevalue>
  </binding>
  <binding>
    <reference>
      <association id="vdoc001" state="id"/>
    </reference>
    <featurevalue feature="direction">dest</featurevalue>
  </binding>
  <context>
    <contextvalue key="time">morning</contextvalue>
  </context>
</association>
<association id="link002">
  <structure>link</structure><feature>direction</feature>
  <description>Night</description>
  <binding missing="variable">
    <featurevalue feature="direction">src</featurevalue>
  </binding>
  <binding>
    <reference>
      <association id="vdoc001" state="id"/>
    </reference>
    <featurevalue feature="direction">dest</featurevalue>
  </binding>
  <context>
    <contextvalue key="time">night</contextvalue>
  </context>
</association>
```

Figure 6: The Sculptural Links corresponding to the Virtual Document in Figure 4

5.1.2 Maintaining Consistent State

When a user follows a link across applications their state remains based on the old application and may have no relevance to the new one. There are two ways to cope with this potential statelessness.

One option is to add details into the script about how to initialise the state of an application given someone arriving from an external, unknown position. This retains the encapsulation of applications

but acknowledges that unexpected arrival is possible. An application might choose to send the visitor to the start of the application, or to set up a midway position as best as it could.

A more complex method would be to use the behaviour objects on the links to record the state that the system should take upon arrival. This also retains the encapsulation of different applications, as each application defines all its own *incoming* links, which are the ones that would need to contain the state information. When a link is followed, and before the new application is opened, the system ensures that the application is initiated for its required successor state.

After incorporating Linky linkbases in AHA! is it possible to design the adaptive system with more advanced means to update the state space. For each state, a user can navigate to a set of page templates. Because an update of the state space is always related to the chosen template, the use of templates is limited: for every state of the system, a user has direct access to zero or one views of a template. In order to navigate with possibilities of multiple views of one template, the hyperlink has to be equipped with specific context adaptations. Linky’s structure allows context and behaviour to be attached to Data objects. With Linky’s hyperlink mechanism, any initialisation of the system and templates can be reached, independent of the current state space.

5.2 Intra-application Linking

By using a separate OHS alongside AHA! it becomes possible to use the individual applications as components of a larger system. The sculptural links connect in a free-form way previously unconnected hypertext networks (inter-application links). However, it also creates new connections within those networks (intra-application links). While this can cause some problems for a tightly designed system, which now has to now deal with navigational choices that are not pre-determined, it also offers significant benefits.

In AHA! an author has to store links manually in the set of templates. While inserting content, each position in the state space where the template could appear has to be considered. Using a linkbase helps authors by encouraging them to declare which links are needed for each state of the system; these intra-application links are then added automatically.

An improvement offered by incorporating a linkbase is that it provides the author with new techniques to update the state space. Previously for each system state and for each template a user had access to only one available view, which is implicit, but determined. With a linkbase, the various free-formed links towards the same template can have different explicit assignment sets and thus each provides a different view of that template.

When applications are constructed for external access, processes beyond the scope of the visited application are allowed to update data elements. To ensure that modifications from an external application are actually improvements, any dependencies between data values should be carefully designed and applications must be able to restore themselves to a desired state.

Due to the tight constraints that govern most AHA! applications it is doubtful that many inter-application links would be generated. Even applications that share a large portion of a subject (such as applications on different types of cookery) will have large condition sets, especially for pages intended for viewing late in the adaptive presentation.

The effectiveness of the sculptural link depends inversely on the ability of the compiler to calculate conditions based on the original scripts. For example, if the compiler is able to gather *all* the conditions correctly than a single application will be identical when

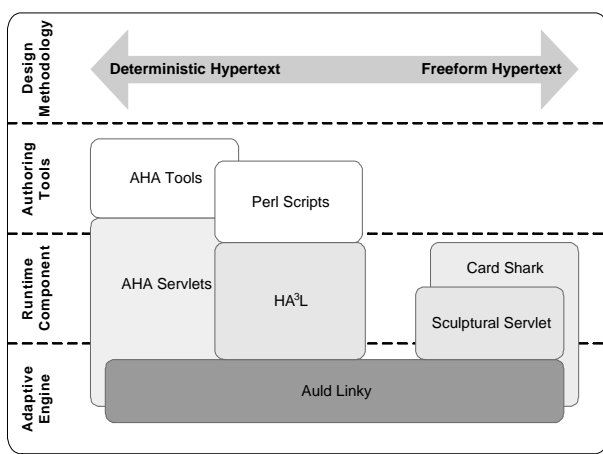


Figure 7: Relationship between Methodologies and Systems

run on either AHA! or the Sculptural Servlet and inter-application linking will probably be very limited.

If the conditions on the sculptural links are looser than the script intended then it is likely that more inter-application links will be formed and also that intra-application links will be created that we're not intended in the original design. It remains to be seen where the balance lies between a few appropriate opportunistic links and a more chaotic tangle.

6. LESSONS LEARNED

Our initial hopes were to draw upon Open Adaptive Hypermedia work and explore how Auld Linky, an Open Hypermedia System, might be integrated with an established Adaptive Hypermedia System such as AHA!. We chose to use Sculptural Hypertext as a way in which AHA! applications might be augmented as well as replicated.

We quickly realised that a major obstacle to this integration were the very different design styles promoted by Adaptive and Sculptural Hypertext. AHA!'s methodology favours a tightly controlled and deterministic design, where the system's behaviour is (ideally) well known. Sculptural Hypertext (and to a lesser extent Open Hypermedia in general) emphasises an extensible hyperstructure, where the author has only limited ability to predict the navigation options available to a reader at any one time.

The experience has led us to question the relationship between the two systems and the forms of hypermedia they promote.

6.1 Designed vs. Freeform Hypermedia

Figure 7 shows how we believe AHA! and Auld Linky, and the methodologies they support, relate to one another. In addition to AHA! and Auld Linky the diagram also shows Bernstein's sculptural system, Card Shark [6] and the Open Adaptive System called HA³L [4].

Card Shark is an application that was designed explicitly to explore a sculptural, authoring methodology. Its run-time component and adaptive engine are combined together (into a Flash application).

Although the Sculptural Servlet doesn't have any tools to help authors create sculptural hypertexts it is built on a common adaptive engine, Auld Linky. Because of this it can take advantage of the reusable, consistent structure and use links that have virtual documents and tours as destinations. It can also cope with multiple links to the same destination, analogous in Card Shark to having several

ways in which to play a card.

HA³L (Hypermedia Adaptation using Agents and Auld Linky) is an agent system that implements an adaptive medical web application using Auld Linky as the adaptive engine, thus demonstrating that Auld Linky is capable of supporting the techniques described in Brusilovsky's taxonomy [11, 12]. HA³L has no direct authoring tools as the hyperstructure it serves was extracted from the XML pages of an existing system using Perl scripts to uncover the implicit relationships and convert them to FOHM structures. The system contains elements of both designed and freeform hypertext but it is less deterministic than AHA! as it allows for additional hyperstructure (such as annotations or user authored links) to be added at runtime.

It is our belief that an OHS which uses context (or one which has an extensible set of behaviours) can support a full spectrum of design methodologies, from very deterministic to very freeform. Such systems do not actually impose any methodological restrictions on their use, although it is understandable that this in itself might be perceived as encouraging less deterministic hypertexts.

What systems such as Linky are missing is the authoring tools to support the various methodologies. We have discussed this problem in regard to Sculptural Hypertext before [7]. With Sculptural Hypertext there is a general issue with generating authoring tools in that the actual methodology is not yet well understood.

This lack of authoring tools is a direct result of the original objective of Auld Linky, which was to explore new domains of hypertext and provide a basis for experimentation. However, it provides a serious obstacle for those people who are interested in one methodology in particular, or who wish to use Auld Linky for serious deployment.

In contrast AHA! specifically promotes the deterministic hypertext methodology and provides a complete set of tools to support this along with the run-time component and adaptive engine needed to drive the final hypertext.

6.2 Improving AHA! and Auld Linky

The difference between the methodologies of AHA! and Sculptural Hypertext means that there can be no tight integration between the two systems without loosening some of their fundamental design principles.

Despite this, understanding the relationship between them is still a constructive step. It indicates that it would be possible to construct a system such as AHA! on top of a common adaptive engine like Auld Linky and, as long as the structures used are common navigational ones (links, virtual documents, etc), generic browsers would be able to render them, moving towards interoperability between hypertext systems.

Our experience of trying to calculate links between different AHA! applications indicates that AHA! might benefit from increasing the semantic description of its pages (using a common vocabulary or even an ontology) so that multiple applications might be managed together more easily.

In return we are encouraged to build up from the OHS layer and examine the various methodologies from a common point of view. Exploring both the type of tools that would be useful and also the type of hypertexts that are best suited to the different design approaches.

Although we have used Auld Linky as our example of an appropriate structure layer, other more general structural systems such as Construct [23] or Callimachus [13] could have contextual or adaptation services defined for them and then be used in the same way.

Figure 8 shows how a structure server such as Auld Linky might be used as the adaptive engine of AHA!. The modelling methodol-

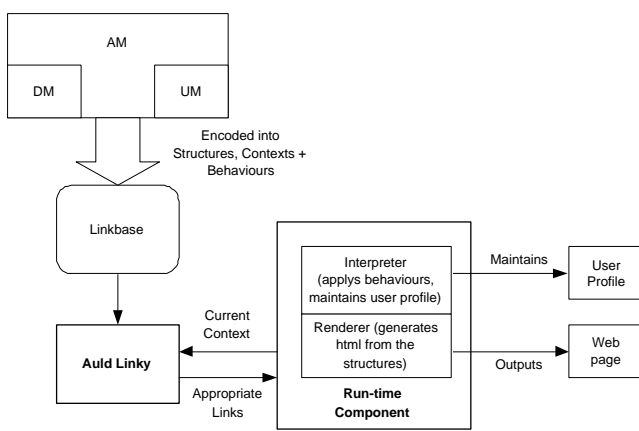


Figure 8: Auld Linky as the Adaptive Engine of AHA!

ogy of AHA! is used to create a linkbase that encodes the concept objects as contexts and behaviours. There is still a need for a conceptual engine to interpret the behaviours and a renderer to create the web pages as output but it is Linky that does all the selection of the hyperstructure.

While this would be possible at present, Linky's limited context object (a anonymous set of attribute/value pairs) would make the linkbase unwieldy as each context object and behaviour would need to be very large.

This shows that any context services that are defined at the structure level, such as Auld Linky's context model and culling process, need to support complex concept networks as these are common in adaptive applications.

7. CONCLUSIONS

In this paper we have presented an initial attempt to integrate AHA! and Auld Linky as a means to explore the relationship between them. Although our implementation has not been completely successful, we have discovered through the design work that this is because of methodological differences between the two systems, a conflict between planned, deterministic hypertexts (AHA!'s adaptive hypermedia) and freeform hypertexts (Auld Linky and Sculptural Hypermedia in particular).

We have also explained that this divide does not necessarily extend into the adaptive engine and that a generic structure server such as a OHS could support both methodologies given the right tools.

There are currently projects underway that are investigating common guidelines, techniques and tools for user modelling and common structures for adaptive hypermedia [14]. The hierarchical concept designs of AHA! and the open extensible contextual structures of Auld Linky represent valuable lessons learned in these area.

Adaptive Hypermedia is maturing as a research area and we are now seeing a relationship emerge between it and other hypermedia approaches. We believe that future systems will be able to use this understanding to exploit the advantages and disadvantages of each approach, as well as model, author and run hypertexts using common tools and components.

8. ACKNOWLEDGEMENTS

We would particularly like to thank Wendy Hall for her efforts towards making this collaboration possible. This research is funded in part by EPSRC IRC project "EQUATOR" GR/N15986/01, by

the AHA! project of the NLnet Foundation, and by the Socrates Minerva project ADAPTS (101144-CP-1-2002-1-NL-MINERVA-M).

9. REFERENCES

- [1] P. Brusilovsky and E. Schwarz and T. Weber. A tool for developing adaptive electronic textbooks on www. In *Proceedings of the AACE WebNet'96 Conference*, pages 64–69, 1996.
- [2] Kenneth M. Anderson, Richard N. Taylor, and E. James Whitehead. Chimera: Hypertext for Heterogeneous Software Environments. In *ECHT '94. Proceedings of the ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK*, pages 94–197, 1994.
- [3] Keith Andrews, Frank Kappe, and Herman Maurer. Serving information to the web with hyper-g. *The Third International World-Wide Web Conference. Darmstadt, Germany*, 27:919–926, 1995. Published in Computer Networks and ISDN Systems.
- [4] Christopher Bailey. *An Agent-Based Framework to Support Adaptive Hypermedia*. PhD thesis, Department of Electronics and Computer Science, University of Southampton, UK, 2002.
- [5] Christopher Bailey, Wendy Hall, David E. Millard, and Mark J. Weal. A Structural Approach to Adaptive Hypermedia. In *Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web Based Systems, Malaga, Spain.*, May 2002.
- [6] Mark Bernstein. Card shark and thespis: exotic tools for hypertext narrative. In *Proceedings of the '01 ACM conference on Hypertext, Aarhus, Denmark*, pages 41–50. ACM Press, 2001.
- [7] Mark Bernstein, Mark J. Weal, and David E. Millard. On Writing Sculptural Hypertext. In *Proceedings of the '01 ACM conference on Hypertext, Aarhus, Denmark*, pages 65–66, 2001.
- [8] P. De Bra and I. Calvi. Aha! an open adaptive hypermedia architecture. *The New Review of Hypertext and Multimedia*, 4:115–139, 1998.
- [9] Paul De Bra, Ad Aerts, and Brendan Rousseau. Concept Relationship Types for AHA! 2.0. In *Proceedings of E-Learn'02, Montreal, Canada, AACE*, 2002.
- [10] P. Brusilovsky, J. Eklund, and E. Schwarz. Web-based education for all: A tool for developing adaptive courseware. In *Computer Networks and ISDN Systems. Proceedings of 7th International World Wide Web Conference, April 14- 18, 30 (1-7)*, pages 291–300, 1998.
- [11] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. In *User Modelling and User-Adapted Interaction : Special Issue on adaptive hypertext and hypermedia*, volume 6, pages 87–129, Berlin/Heidelberg/New York, 1996. Kluwer academic publishers.
- [12] Peter Brusilovsky. Adaptive hypermedia. *User Modeling and User-Adapted Interaction, Ten Year Anniversary Issue*, 11:87–110, 2001.
- [13] Dimitris Christodoulakis, Michael Vaitis, Athanasios Papadopoulas, and Manolis Tzagarakis. The callimachus approach to distributed hypermedia. pages 47–48, 1999.
- [14] A.I. Cristea and P. De Bra. ODL Education Environments based on Adaptivity and Adaptability. In *Proceedings of E-Learn'02, Montreal, Canada, AACE*, 2002.

- [15] Andrew M. Fountain, Wendy Hall, Ian Heath, and Hugh C. Davis. MICROCOSM: An Open Model for Hypermedia With Dynamic Linking. In A. Rizk, N. Streitz, and J. André, editors, *Hypertext: Concepts, Systems and Applications (Proceedings of ECHT'90)*, pages 298–311. Cambridge University Press, 1990.
- [16] Kaj Grønbaek and Randall H. Trigg. Design issues for a Dexter-based hypermedia system. *Communications of the ACM*, 37(3):40–49, February 1994.
- [17] Norman Meyrowitz. Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework. In *OOPSLA '86. Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 186–201, September 1986.
- [18] Danius T. Michaelides, David E. Millard, Mark J. Weal, and David C. De Roure. Auld leaky: A contextual open hypermedia link server. In Siegfried Reich and Kenneth M. Anderson, editors, *OHS7, SC3 and AH3, Proceedings of the ...*, Published in *Lecture Notes in Computer Science, (LNCS 2266)*, Springer Verlag, Heidelberg (ISSN 0302-9743), pages 59–70, 2001.
- [19] David E. Millard, Luc Moreau, Hugh C. Davis, and Siegfried Reich. FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability Between Hypertext Domains. In *Proceedings of the '00 ACM Conference on Hypertext, May 30 - June 3, San Antonio, TX*, pages 93–102, 2000.
- [20] Wolfgang Nejdl and Martin Wolpers. Kbs hyperbook - a data-driven information system on the web. In *Proceedings of the The Eighth International World Wide Web Conference, Toronto, Canada, 1999*.
- [21] John L. Schnase, John L. Leggett, David L. Hicks, Peter J. Nuernberg, and J. Alfredo Sánchez. Design and implementation of the HB1 hyperbase management system. *Electronic Publishing—Origination Dissemination and Design*, 6(1):35–63, June 1993.
- [22] Mark J. Weal, David E. Millard, Danius T. Michaelides, and David C. De Roure. Building Narrative Structures Using Context Based Linking. In *Proceedings of the '02 ACM conference on Hypertext, Maryland, U.S.A.*, pages 37–38, 2002.
- [23] Uffe Kock Wiil and Peter J. Nürnberg. Evolving Hypermedia Middleware Services: Lessons and Observations. In *Proceedings of the 1999 ACM Symposium on Applied Computing (SAC '99), San Antonio, TX*, pages 427–436, February 1999.