AHA! Version 2.0
More Adaptation Flexibility for Authors

Paul De Bra*, Ad Aerts, David Smits, Natalia Stash
Department of Computing Science
Eindhoven University of Technology (TUE)
PO Box 513, Eindhoven, The Netherlands
debra@win.tue.nl

Abstract: AHA! is a simple Web-based adaptive hypermedia system. Because of this simplicity it has been studied and experimented with in several research groups, see e.g. (Cini & Valdeni de Lima, 2002), (Calvi & Cristea, 2002), (Romero et al., 2002). This paper identifies shortcomings in AHA! and presents AHA! version 2.0 which tries to overcome the known problems with AHA! while maintaining its biggest asset: simplicity. We illustrate how different user modeling and adaptation aspects can be expressed in the new AHA! system (that would be difficult in the old system, and impossible in most other systems).

Introduction and Background

The development of the AHA! system started in 1996, when we created server-side software (CGI-scripts at that time) to add adaptation to an on-line course text on the subject of hypermedia. (For an introduction in the field of adaptive hypermedia, see (Brusilovsky, 2001). The AHA! software has been changed and extended somewhat over the years, resulting in what we will refer to as AHA! Version 1.0 in this paper, and which is described e.g. in (De Bra et al., 2000) and (De Bra & Ruiter, 2001). AHA! 1.0 is based on a simple architecture for an adaptive Web-based system:

- An application consists of a number of *concepts*. Some concepts represent Web-pages while others represent higher-level or abstract entities.
- The user model (for each user) consists of a *value* for each *concept*. The most common interpretation of this value is that the value means the *knowledge level* of the user with respect to the concept.
- Each time a user visits a page the *value* corresponding to that page is *increased*. In AHA! the author can define propagation rules that cause this increase to influence the value for other concepts. It is thus easy to define a structure of pages, sections and chapters, meaning that reading a page contributes to the knowledge of a section, and a knowledge increase for a section implies a (smaller) knowledge increase in the chapter. The propagated changes are arbitrary. It is also possible to register a *decrease* in knowledge of a concept by accessing a certain page.
- The author can define *requirements* for fragments of a page and for whole pages. Requirements are Boolean expressions over knowledge values for concepts. When the requirement for a fragment of the page to be presented is *true* the fragment is included in the presentation. (Otherwise the fragment is left out.) When the requirement for a page is *true* all anchors of links to this page are shown in a clearly visible color (typically blue or purple in AHA!). Otherwise the anchors are shown in *black*, meaning that they will be hidden in the text which is also black. (The black links are also *not underlined* to make them effectively hidden.)

As several researchers have pointed out, including (Calvi & Cristea, 2002), a consequence of the simplicity of AHA! is that it is only easy to create simple, straightforward applications. In this sense AHA! is in essence not very different from other adaptive systems like Interbook (Brusilovsky et al, 1998) which make it easy to develop one type of application, namely on-line courses. In (De Bra et al., 2000) we showed that it *is* possible to develop different types of applications in AHA! or to base adaptation on other user or browsing aspects than just knowledge. However, the examples that were given also showed that this development is difficult, and it was very clear that the user modeling and adaptation features of AHA! had to be "abused" in order to realize these alternative applications.

* Also at the "Centrum voor Wiskunde en Informatica" in Amsterdam, and the University of Antwerp, Belgium.

In AHA! Version 2.0 we address this problem through a richer user model structure and a domain and adaptation model with more advanced *requirement* and *generate* rules. The new features are based on the AHAM reference model (De Bra et al., 1999), which is an extension of the well-known Dexter reference model for hypermedia (Halasz & Schwartz, 1994).

The Domain Model / Adaptation Model in AHA! Version 2.0

AHA! deviates from the AHAM referemce model (De Bra et al., 1999) by not separating the *domain model* from the *adaptation model*. In AHA! the author defines *concepts*, along with *requirements* that determine under which conditions the user is "ready" to access the concept, and *generate rules* that specify how the browsing behavior of the user translates into user model updates. The *generate rules* are *condition-action rules* as used in active database theory (Baralis & Widom, 2000). We illustrate the AHA! concept structure with a continuation of the beer and chocolate example of (De Bra & Ruiter, 2001). AHA! 2.0 supports storing the concept structure in XML files or in a mySQL database. We use the XML representation in this paper. (There are some minor deviations between the syntax in this paper and the actual AHA! 2.0 syntax, for readability and simplicity reasons.)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE conceptList SYSTEM 'concept.dtd'>
<conceptList>
<name>Belgian Beer Example</name>
<concept>
  <name>de_koninck</name>
  <desc>first author's favorite beer</desc>
  <resource>de_koninck.xhtml</resource>
  <req>beer.interest > 50</req>
  <attribute name="access" type="bool"
    isPersistent="false" isSystem="true"
    isChangeable="false">
    <default>false</default>
    <generateListItem isPropagating="true">
      <req>beer.interest < 100</req>
      <trueActions>
        <action>
          <concept>beer</concept>
          <attribute>interest</attribute>
          <expr>beer.interest + 10</expr>
        </action>
      </trueActions>
    </generateListItem>
    <generateListItem isPropagating="true">
      <req>chocolate.interest < 50 and
           chocolate.interest > 4</req>
      <trueActions>
        <action>
          <concept>chocolate</concept>
          <attribute>interest</attribute>
          <expr>chocolate.interest-5</expr>
        </action>
      </trueActions>
    </generateListItem>
    <generateListItem isPropagating="true">
      <req>beer.interest > 50</req>
      <trueActions>
        <action>
          <concept>de_koninck</concept>
          <attribute>knowledge</attribute>
          <expr>100</expr>
        </action>
      </trueActions>
      <falseActions>
        <action>
          <concept>de_konink</concept>
          <attribute>knowledge</attribute>
          <expr>35</expr>
        </action>
      </falseActions>
    </generateListItem>
  </attribute>
  <attribute name="knowledge" type="bool"
    isPersistent="true" isSystem="false"
    isChangeable="true">
    <default>0</default>
    <generateListItem isPropagating="true">
      <req>true</req>
      <trueActions>
        <action>
          <concept>beer</concept>
          <attribute>knowledge</attribute>
          <expr>beer.knowledge + 0.2 *
            _de_koninck.knowledge</expr>
        </action>
      </trueActions>
    </generateListItem>
  </attribute>
</concept>
<concept>
  ... next concept definition here ...
</concept>
  ... more concept definitions ...
</conceptList>
```

The above concept definition is linked to the page "de_koninck.xhtml". This page is considered "desirable" by the system if the requirement "`beer.interest > 50`" is fulfilled. For adaptation this means that links to the de_koninck page will have a blue or purple anchor only for users with a high interest in beer. (The link will be blue before the first visit to the page and purple afterwards.) Uninterested users will not easily see the links to a special

beer like "De Koninck" because the link anchors are hidden by making them black like normal text, and not underlined. When a user accesses the de_koninck page the attribute `access` *temporarily* becomes true. The attribute is a "system" attribute, meaning its value is changed by a system even, in this case the access of the page. The attribute is not persistent, meaning that its value returns to the default value after dealing with the access event. The event causes the three actions (or "generateListItem"s) associated with this attribute to be considered for execution. The first action increases the interest in beer, but only if this interest is not yet 100 or more. The second action decreases the interest in chocolate, but only if this interest is not yet known to be high (but high enough to not become negative after the decrease). The third action registers that for interested users the knowledge about de_koninck becomes 100 and for uninterested users (who happen to find the page even though the links to it are hidden) the knowledge becomes only 35. "`knowledge`" is a persistent attribute of de_koninck. meaning that it is stored in the user model. Initially the attribute has the value 0 (the "default"). The rule associated with the knowledge attribute means that any change to this knowledge affects the knowledge of the higher level concept "`beer`" as well. 20% of the *change in the knowledge* of de_koninck is transferred to "`beer`". The "_" is used to indicate that the *change* in the attribute value is used, not the *value* of the attribute itself. As one can see from the example, *accessing* the de_koninck page changes the *interest* in beer and in chocolate and raises the *knowledge* of de_koninck; this knowledge change triggers the rule for the knowledge attribute of de_koninck, which will then raise the *knowledge* of beer. The change in interest for beer and chocolate and the change in knowledge of beer may also trigger rules associated with these concepts and attributes but that cannot be seen from the example as the concept structure of beer and chocolate is not given.

The above example may seem excessively verbose for expressing a few fairly simple rules for updating the user model. The given XML file only shows a single concept, whereas even a small application will already consist of 100 concepts or more. Fortunately a simple (Java applet based) authoring interface hides this verbose syntax from the author. Also, most applications require only a few simple types of rules, like having the access of a page raise knowledge, and like having knowledge propagate through a concept hierarchy.

Adaptive applications are sometimes concerned with different aspects of users, including *knowledge*, *interests*, *goals*, *learning style*, etc., and they may also wish to perform adaptation to other circumstances, like differences in browser, window size, network bandwidth, location, the user's background and Web experience, etc. The big advantage of AHA! 2.0 over its predecessor is that each aspect can be represented through different concepts or through different attributes of concepts. This makes it possible to clearly distinguish adaptation aspects that are unrelated to each other. The use of attributes also makes it possible to "combine" related adaptation aspects such as *knowledge about* a concept and *interest in* that same concept.

AHA! allows each concept to have a different set of attributes associated with it, and different condition-action rules. Attributes can be of type Boolean, integer or string. Each rule execution can trigger other rules, as seen in the example above. It is difficult to predict in general how the rule engine will behave as a whole. We have studied the problems of *termination* and *confluence* in the past (Wu et al., 2001). However, we believe the easiest way to enable authors to oversee the potential problems is to have an "`isPropagating`" attribute per action that indicates whether this action is allowed to trigger other actions or not. This was also proposed in the AHAM model (De Bra et al., 1999). As the action indicates (in the assignment) which attribute of which concept is updated, it is immediately clear which rule propagation will occur. As in all such rule systems (Baralis & Widom, 2000) there is a danger that the rule execution through this propagation mechanism will not terminate. In the previous version of AHA! an arbitrary propagation constraint was introduced (De Bra et al, 2000) that guaranteed termination. Unfortunately the constraint could also be a source of results the author did not anticipate. In AHA! 2.0 this constraint is lifted. Rules are always allowed to trigger each other, provided that the "`isPropagating`" attributes are set to true. Furthermore, the author can control more precisely the conditions under which a rule is allowed to be executed. In (Wu et al, 2001) we indicate how potential infinite loops can be detected during the authoring process.

In Figure 1 we show a two screendumps of the Java-based authoring tool for entering the concept structure, and the rules for updating the user model that is used for creating the adaptive presentation. The first image shows that the concept "`de_koninck`" has attributes "`access`" and "`knowledge`", and that three actions or generateListItems are associated with this attribute. The interface shows the condition (`<req>`) of each rule. The second image shows how the third generateListItem is defined. While entering expressions (like "`beer.interest > 50`") the tool shows the expression in red until it is syntactically correct and uses only concepts and attributes that have been previously defined. This guarantees that the complete structure of concepts, attributes and generateListItems is sound.

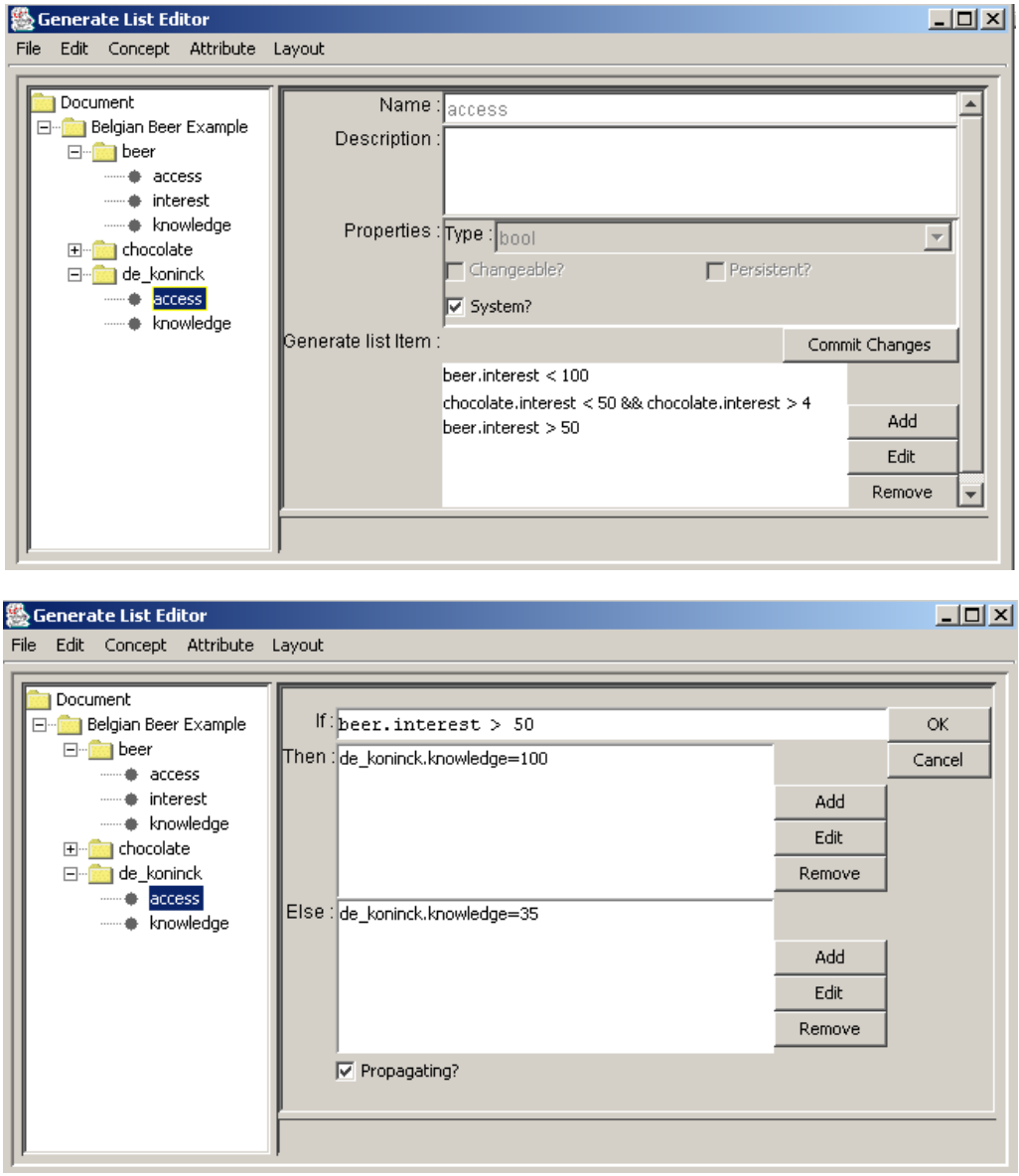

Figure 1: AHA 2.0 authoring tool

The User Model in AHA! Version 2.0

The most common type of user model in adaptive hypermedia applications (Brusilovsky, 2001) is the *overlay model*. For every concept and attribute in the domain model of the application a concept and attribute is kept in the user model. AHA! 2.0 follows the same philosophy, albeit with two exceptions:

- Attributes (of concepts) in the domain/adaptation model can be defined *not* to be *persistent*. The `access` attribute is a typical one. When the *adaptation engine* runs (when a page is accessed) these volatile attributes are initialized using the "default" value, and any updates to these values are lost when the engine terminates. The access attribute for instance is *false* for all concepts, but it will be temporarily true for the concept (page) that is accessed.
- When a user logs in for the first time a special "concept" is created, called `personal`, which is used to store user-related aspects that are not about the subject domain of the application. This information includes a name, email, login-id, password, and any other information the application asks for during the

login process. For our on-line course on hypermedia this information also includes the student's university and major. Because this information is represented through a concept it can be used for adaptation. We can thus present different assignments to students from different universities for instance. (In AHA! 1.0 the login information was not available for adaptation purposes.)

Below we show a small part of a user model for the imaginary beer application.

```
<!DOCTYPE profile SYSTEM 'profile.dtd'>
<profile>
  <record>
    <key>personal.id</key>
    <type>string</type>
    <persistent>true</persistent>
    <value>debra</value>
  </record>
  <record>
    <key>personal.email</key>
    <type>string</type>
    <persistent>true</persistent>
    <value>debra@win.tue.nl</value>
  </record>
  <record>
    <key>personal.goodlink</key>
    <type>string</type>
    <persistent>true</persistent>
    <value>0000ff</value>
  </record>
  <record>
    <key>personal.badlink</key>
    <type>string</type>
    <persistent>true</persistent>
    <value>000000</value>
  </record>
  ...
  <record>
    <key>de_koninck.knowledge</key>
    <type>int</type>
    <persistent>true</persistent>
    <value>35</value>
  </record>
  <record>
    <key>beer.knowledge</key>
    <type>int</type>
    <persistent>true</persistent>
    <value>20</value>
  </record>
  <record>
    <key>beer.interest</key>
    <type>int</type>
    <persistent>true</persistent>
    <value>40</value>
  </record>
  <record>
    <key>chocolate.knowledge</key>
    <type>int</type>
    <persistent>true</persistent>
    <value>15</value>
  </record>
  <record>
    <key>chocolate.interest</key>
    <type>int</type>
    <persistent>true</persistent>
    <value>20</value>
  </record>
  ...
</profile>
```

Document (Page) Formats

AHA! applications consist of sets of *pages* with *links* between them. Currently (in both version 1.0 and 2.0) the basic format is HTML, because that is what browsers understand. In AHA! 2.0 there are three "variants" in which source pages can be written (and which AHA! translates to plain HTML before sending it to the browser):

- There are some XML tags that have a special meaning for AHA!. The first AHA! format consists of XML pages with these tags, combined with HTML. The most important AHA! tags are the <if> tag for the conditional inclusion of fragments, and the <a> tag for links. The output of AHA! is the HTML source page in which some fragments may be deleted, and in which the links are shown in different colors depending on the "desirability" of the page that is the link destination.
- Plain HTML documents can also be used with AHA!. In such documents links are adapted like in the first format. While HTML documents do not take full advantage of the possibilities of AHA! it is important to allow them in order to include *external* documents in an adaptive application. (AHA! 2.0 can serve pages it first retrieves from other servers.)
- Using *modularized DTDs*, (and the Xerces XML parser that supports them) pages can now also be written in a format that combines AHA! tags with XHTML. The AHA! engine only looks at the AHA! tags to perform its adaptation. As a result it will be easy to extend AHA! with the capability to serve other

document formats combined with AHA! tags as well. (SMIL is one of the formats we wish to experiment with in the future.)

AHA! does not enforce or provide any special page formats. We are using AHA! in an on-line course without using HTML frames, and in another course with frames, and also with Javascript.

Examples of AHA! applications

The use of AHA! for modeling and adapting to *knowledge* and *interests* has been shown in previous sections. The versatility of AHA! becomes apparent when one looks at the following two small illustrations of completely different adaptation:

- In one on-line course we wished to have two presentations: a terse one (like viewgraphs) and a verbose one (like full course text). By letting the user set a *verbosity* level (which can be implemented as an attribute of the "personal" concept) we could have the system select fragments to show and hide. Currently we provide only two levels (terse of verbose) in that course, but by using an integer attribute we could also introduce more levels, showing more and more details and longer explanations as the verbosity level is cranked up.
- Our course that uses frames has a "navigation bar" in the left frame, and shows the pages in the (larger) right frame. The navigation bar shows course topics, and can optionally show a submenu for each topic. When a page on a certain topic is shown the corresponding menu should be opened and other menus should be closed. We realized this by turning all the menus into fragments to be conditionally included. Each time a page is accessed it activates the corresponding menu. The activation is done through a one line Javascript program that tells the browser to reload the navigation bar. There are two ways to implement the menu system in AHA!. One way is to create a "menu" concept with an Boolean attribute for each menu. Each page access triggers a set of rules that open up the appropriate menu(s) and close the others. It is possible in this scheme to leave more than one menu open. A drawback of this solution is that when a new topic is added to the course a large number of rules must be added to close the menu for that topic whenever any page from any other topic is accessed. A second way is to create an integer or string menu attribute (for either the personal concept or a menu concept). Each menu is assigned a number or name, and each page sets the menu attribute to the number or name of the menu to be opened. This solution does not have a problem when new topics are added. A drawback of this solution however is that only one menu can be open at once.

Adaptation features like these two could already be expressed in AHA! 1.0 but that required playing tricks with the concepts and adaptation rules. In AHA! 2.0 one can define concepts and attributes with a specific special purpose in mind and the associated rules cannot be easily "mixed up" with concepts and attributes that are used to represent knowledge and interest. We consider the ability to clearly distinguish different subparts of the domain/adaptation model, each having a different purpose, to be the biggest advantage of our new AHA! version.

Conclusions and Future Work

With AHA! 2.0 we have provided a versatile user modeling and adaptation tool. We hope that AHA! will be suitable for many different types of applications. Our experience with using AHA! for on-line courses has shown that in addition to modeling knowledge and providing adaptation based on it, AHA! can also be used to provide adaptation to personal aspects of a user, including preferences (like color or verbose vs. terse text).

The presentation format in AHA! is completely up to the author. The structure with concepts and attributes can also be used to help in creating the presentation. We for instance showed how to create a navigation frame with menu and submenus.

While we are still gaining experience in using AHA! 2.0 we already have a wish-list for a future version. Linking in AHA! is page-based. We plan to allow links to a higher level concept in the future, along with the automatic selection of the best page to be shown when a user "clicks" on such a link. We also plan to allow more flexible page construction. Currently a page consists of a sequence of fragments that are conditionally included, in a fixed order only. For information retrieval applications we need to allow at least the sorting of fragments.

Acknowledgement

The development of AHA! was made possible through a grant of the NLnet Foundation. Apart from them we would also like to thank the many researchers who have studied our previous research and development on AHA! and some of which have even developed applications with AHA!. Their research and experience reports have been very valuable in deciding how to extend and improve AHA! so that more people will become able to use it to develop adaptive courses and possibly also other adaptive applications.

References

Baralis, E., Widom, J. (2000). *An algebraic approach to static analysis of active database rules*. ACM Transactions on Database Systems, 20:1, pp. 269-332.

Brusilovsky, P. (2001). *Adaptive hypermedia*. User Modeling and User Adapted Interaction, 11 (1/2) pp. 87-110.

Brusilovsky, P., Eklund, J., Schwarz, E. (1998). *Web-based Education for All: A Tool for Developing Adaptive Courseware*. Computer Networks and ISDN Systems (Seventh International World Wide Web Conference), 30, 1-7, 291-300.

Calvi, L., Cristea, A. (2002). *Towards Generic Adaptive Systems: Analysis of a Case Study*. Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Springer Verlag, LNCS 2347, pp. 77-87.

Cini, A., Valdeni de Lima, J. (2002). *Adaptivity Conditions Evaluation for the User of Hypermedia Presentations Built with AHA!*. Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Springer Verlag, LNCS 2347, pp. 490-493.

De Bra, P., Brusilovsky, P., Houben, G.J. (1999). *Adaptive Hypermedia: From Systems to Framework*. ACM Computing Surveys 31:4. (URL: http://www.cs.brown.edu/memex/ACM_HypertextTestbed/papers/25.html)

De Bra, P. & Calvi, L. (1998). *AHA: a Generic Adaptive Hypermedia System*. 2nd Workshop on Adaptive Hypertext and Hypermedia, pp. 1-10. (URL: http://wwwis.win.tue.nl/ah98/DeBra.html)

De Bra, P., Aerts, A., Houben, G.J., Wu, H. (2000). *Making General-Purpose Adaptive Hypermedia Work*. Proceedings of the AACE WebNet 2000 Conference, pp. 117-123.

De Bra, P., Ruiter, J.P. (2001). *AHA! Adaptive Hypermedia for All.* Proceedings of the AACE WebNet Conference, pp. 262-268.

De Bra, P., Houben, G.J. & Wu, H. (1999). *AHAM, A Dexter-based Reference Model for Adaptive Hypermedia*, Proceedings of the ACM Conference on Hypertext and Hypermedia, pp. 147-156.

Halasz, F., Schwartz, M. (1994). *The Dexter Hypertext Reference Model*. Communications of the ACM, 37:2, pp. 30-39.

Romero, C., De Bra, P., Ventura, S., de Castro, C. (2002). *Using Knowledge Levels with AHA! for Discovering Interesting Relationships*. Proceedings of the AACE ELearn'2002 Conference.

Wu, H., De Kort, E., De Bra, P. (2001). *Design Issues for General-Purpose Adaptive Hypermedia Systems*. Proceedings of the ACM Conference on Hypertext and Hypermedia, pp. 141-150.