Rapport Interne n° 236, Laboratoire d'Informatique, Université de Tours, octobre 2000.

A web-based approach to adaptive information structuring

M. Crucianu, E. Martin, G. Verley
Laboratoire Informatique, Université de Tours
Ecole d'Ingénieurs en Informatique pour l'Industrie
64, avenue Jean Portalis, 37200 Tours, FRANCE
{crucianu, verley}@univ-tours.fr
http://www.rfai.li.univ-tours.fr/

Abstract: We focus here on a single aspect of adaptive hypermedia, namely the adaptive structuring of information, and get into details regarding the characteristics of the system we developed and the experiments we performed. We end by discussing possible extensions of our system, and outline existing and new directions in the development of adaptive hypermedia.

Keywords: adaptive hypermedia, information structuring, knowledge management.

1 Introduction

Multimedia systems allow users to consult and manipulate different types of media, such as text, sound, video, computer graphics and animation. When several multimedia documents also contain explicit links to each other, one can speak of hypermedia. The increasing popularity of the World Wide Web highlights the growing importance of hypermedia systems for every domain of human activity. While originally developed as tools for user-driven and interactive access to information, hypermedia systems proved to be the enabling technology of a fast and pervasive evolution. However, by giving general access to a huge mass of information, this technology also reinforces and generalizes the difficulties related to the informational overload. The paradoxical but commonly recognized consequence of such an overload is the underuse of information.

Such difficulties can only be alleviated by the development of efficient information management systems, able to provide the appropriate information at the right moment and in a suitable shape. Adaptation is one of the mechanisms which should allow hypermedia systems to meet the abovementioned challenge.

Work on adaptive hypermedia systems began in the early days of the WWW (see [Brusilovsky, 1996] for a comprehensive review of this early work) and was mainly motivated by the development of adaptive online training environments (see e.g. [De Bra and Calvi, 1998]).

Our purpose here is to put forward an approach to the development of specific adaptive hypermedia systems. We then outline several promising research directions in this domain. A more comprehensive discussion of recent developments is left for a later account.

2 Adaptive information structuring and knowledge management

The work presented in this report is part of an on-going project seeking to provide computer assistance for the construction and management of complex knowledge bases. Here, we are attempting to automatically adapt the structure of the knowledge base to changes in its content and to evolutions in its usage. The knowledge base is implemented as a fine-grained database which interacts with the users through an HTTP server. Such bases are usually employed for

the sharing of distributed knowledge across an entire organization, for distance learning and, increasingly, for common interest groups on the Web.

2.1 General ideas

The difficulties encountered by the managers and the users of large knowledge bases implemented as (static or dynamic) Web sites highlight the need for such adaptive behaviour. Indeed, for most bases, there are significant differences between their original design and their usage. These differences usually increase with changes in the contents of the base and the evolution of its use. The structure of the knowledge base becomes quickly outdated. The users have trouble finding and putting together the information they need. The manager of the knowledge base can no longer make sense of the paths followed by the users. Even when good mining tools are available, manually changing the structure of the base in accordance with the results produced by these tools can be very time consuming.

As shown by various experiments, global searches are not **the** solution, mainly because it is very difficult to control the accuracy and the relevance of the results. But global search mechanisms also evolve and, interestingly, tend to become adaptive. Note also that exhaustive search must sometimes be present as a guarantee for completeness.

Our approach here is to assemble a knowledge base out of fine-grained information bricks (we shall call them k-bits and consider they contain no internal link structure), and let its usage adaptively build up and modify a structure, starting from an initial design. A software changes the structure of the knowledge base, structure which is implemented by the interface between the database and the HTTP server. These changes are performed periodically and are based upon the information collected by the server (the logfiles of the HTTP server in the current implementation). We are not concerned here with the adaptation of the base to individual users, but we focus instead on global changes from which every (or most) user(s) can benefit.

Note that the decision to make changes only to the interface between the database and the Web server is limiting in general, but this effect is very low if the database contains only collections of k-bits, as the k-bits lack internal link structure.

Among the many structural changes one can think of, we decided to focus on the following ones:

1° Creating short cuts: the purpose is to selectively diminish the length of the paths the users must follow in order to get the information they need. The decision to create short cuts is based on the time interval between two successive queries and also depends on the local structure of the graph of queries.

2° Aggregating fine-grained information bricks: the purpose is to provide at once all (or at least a big chunk of) the information a user would otherwise retrieve in several queries. By using a fine-grained database one avoids finding overwhelming, loosely related details in the answer to a query. Also, fine grains can be readily reused or discarded. However, aggregating these fine grains can be very time-consuming, and having a software assistance here really helps. The decision to aggregate is based on the presence of strong correlations between queries and makes also use of the time stamps.

3° Adding links for frequent searches: the purpose is to make frequent queries explicit. When the user doesn't see an explicit link to the information of interest, he will make use of the keyword-based search mechanism provided by the site. However, there are usually several difficulties associated to the search (keywords, relevance and accuracy of the results). Corresponding to the most frequent searches, we are simply adding a

query which brings only a selection of the results. This selection is based on a measure of popularity of the answers returned by the search mechanism. When a frequent search gives no result (or very few, according to some measure to be defined), the system can trigger specific requests to content providers/developers.

These changes are well adapted to our use of a fine-grained database, complement each other and achieve a good level of completeness with regard to our purpose.

All the previous work concerning the offline global improvement of a Web site, as opposed to dynamic (within a session or between sessions) adaptation to individual users, is related to our work. We can mention here the WebWatcher in [Armstrong et al., 1995], the work on adaptive web sites in [Perkowitz and Etzioni, 1998] or the "footprints" in [Wexelblat and Maes, 1997]. However, in our work the site is dynamic (based on a database) instead of being just a collection of static pages. Then, our choice of the changes to be applied is new, and coherent with the declared purpose of adaptively evolving a knowledge base out of a fine-grained database. Also, the outcome of the changes we retained here is more than just providing navigation assistance, as these changes concern the very way the available information is structured.

Note that the structure of the knowledge base continuously evolves, trying to follow changes in the contents of the base and the evolution in the ways it is used. This adaptation process obviously has a moving target.

2.2 Implementation issues

Here, we are attempting to automatically adapt the structure of a knowledge base to changes in its content and to evolutions in its usage. The knowledge base is implemented as a fine-grained database which interacts with the environment through an HTTP server.

In our current implementation, the adaptive changes performed by the software to the structure of the knowledge base do not affect the database itself, but are instead limited to the interface between the database and the HTTP server.

For the current implementation we decided to let the software act directly instead of just making suggestions to the manager of the base. To avoid important disruptions of the existing structure during the test period, the software is simply not allowed to make deletions. For the final implementation, however, the best choice would probably be a combination of automatic actions and assistance to the manager of the base.

Our current (toy) implementation makes use of the Internet Database Connector, an IIE server, ODBC and Access on a Windows NT platform. We made these choices simply because the testbeds we employed were based on these tools. A migration to PHP, MySQL and Apache on a Linux platform is under study.

The first implementation was applied to a site developed for the comparison of learning algorithms (called “Validation” in the following) and to an educational site concerning computer peripherals (“Webring” in the following). Please note that this report presents work in progress, so the specifications of the algorithms described here may change in the latest implementation.

Here is the sequence of operations performed for the creation of short cuts and aggregates: build a graph representation for the entire site, select the relevant lines in the logfiles of the HTTP server, identify the user sessions, build the trees corresponding to these sessions and associate time intervals to individual pages, perform some statistical computation, select the short cuts and the aggregates, modify the required files of the original site. Details are given in the following sections.

2.2.1 Rules for the creation of short cuts and aggregates

Before presenting the details of the operations taking place, let us remind the principle of short cuts and aggregates and figure out what kind of information we need.

2.2.1.1 Short cuts

Short cuts have to selectively diminish the length of the paths the users must follow in order to get the information they need. If the users spend some time reading a page (hereafter called "target") — this arguably means they find useful information there — but very little on some previous pages (hereafter called "cut"), then a direct link to the page of interest should be added to the starting page (hereafter called "host"). As an example, if the chain …[page 1] $\xrightarrow{t1}$ [page 2] $\xrightarrow{t2}$ [page 3] $\xrightarrow{t3}$ [page 4] $\xrightarrow{t4}$ [page 5]… appears very often and if the time spent on [page 1] and [page 4] (t1, t4) is usually high, while the time spent on [page 2] and [page 3] (t2, t3) is very low, then a link to [page 4] (the target) is added to [page 1] (the host), right before the link to [page 2] (cut, together with [page 3]). The label and the content of this new link is the same as the one of the link [page 3]→[page 4]. All of the pages mentioned here can be either static or dynamic.

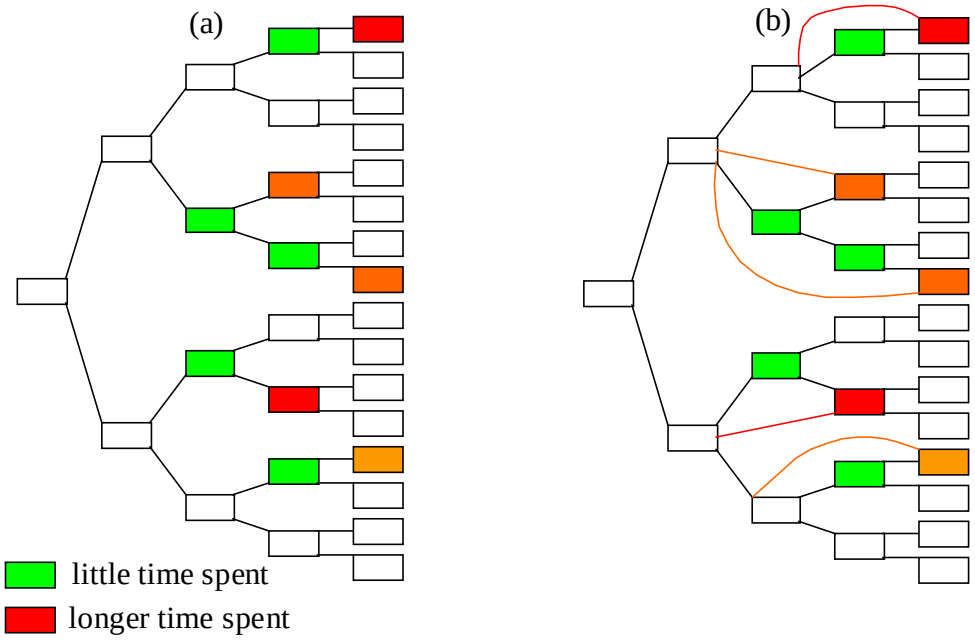

Figure 1: An example of the creation of short cuts. The original architecture of the site is shown in (a), together with an indication of the time spent on the pages which are frequently retrieved. In (b) several short cuts were created by the addition of bypass links.

The short cuts are then based on two important assumptions:

1° The longer the time spent by the users reading a page, the more useful the page is. There is, however, at least one important exception: while pivot pages (containing almost exclusively links, such as the front page of a portal) are usually useful, frequent users spend only a few seconds looking for the link they are interested in. But we believe that short cuts can also be interesting in such cases and, anyway, in the current approach we only add new links and never remove existing ones, so all the pages we "bypass" remain available.

2° By processing the logfiles we can find reliable estimates for the mean time spent by the users on the pages they retrieve. As we shall see later, this is not always the case and

we may need some prior information concerning the web site in order to render such estimates reliable.

The information required for taking the decision to create a short cut and, then, for really creating it, consists of:

1° The (relative) frequency of the presence of every chain in the sessions of different users.

2° The mean time spent by the users on every page.

3° The identity of the host page ([page 1] in the example) and the position of the first link in the original chain ([page 1]→[page 2] in the example; the new link will be placed right before this link).

4° The content and the label of the new link, both coming from the last of the pages which were bypassed ([page 3] in the example).

Note that by taking into account only the mean values for the time spent by the users on every page we may find ourselves in a situation where some of the variances on a chain are so huge that the mean values become irrelevant. No short cuts should be created in such cases. Even when the mean values are relevant, the changes we perform may be quite unsatisfying for a few users, but the adaptation to specific users or groups of users is beyond the scope of this work.

2.2.1.2 Aggregates

The purpose of aggregating fine-grained information bricks is to provide at once all (or at least a big chunk of) the information a user would otherwise retrieve by several queries. Indeed, by using a fine-grained database one avoids finding overwhelming, loosely related details in the answer to a query. Also, fine grains can be readily reused or discarded.

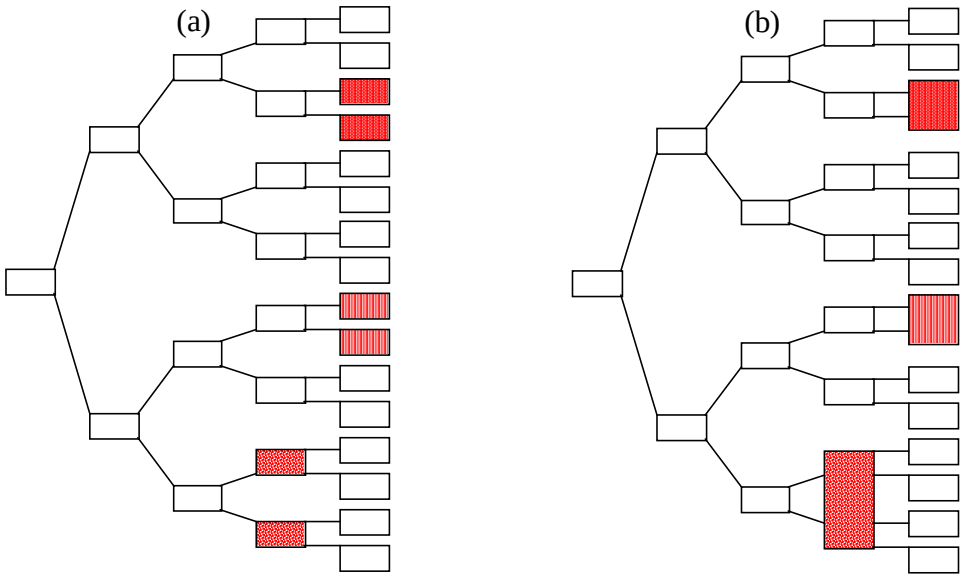

Figure 2: An example of the creation of aggregates. The original architecture of the site is shown in (a), together with an indication of the correlation between queries. In (b) several aggregates were created.

But putting together all the information a user needs can be fairly time consuming. If many users accessed the same information bricks through queries launched from a given page, it would probably be helpful for many other users to find all the information corresponding to these bricks as an answer to a single query. While the database still contains fine-grained bricks, an aggregate query is created and returns an aggregate answer. As an example, if when on [page 1] many users retrieve [page 2] to [page 5], maybe not all of them in the same

sequence, it would probably prove helpful if an aggregate query returning [page 2, page 3, page 4, page 5] were added as a link on [page 1], right above the links to the individual pages 2 to 5.

The creation of aggregates relies then on the following important assumptions:

1° When building aggregates we can avoid overloading the user with poorly structured information. This will probably be the case if the database contains fine-grained information bricks and if we limit the number of bricks in an aggregate.

2° The frequency with which several pages having a common parent appear together in user sessions is a good measure for the degree to which these pages complement each other.

3° By processing the logfiles we can reliably reconstruct the user sessions and find the frequency of retrieval of different pages having a common parent. As we shall see later, this is not always the case and further prior information concerning the web site may be required.

The information required for taking the decision to create an aggregate and, then, for really creating it, consists of:

1° The joint frequency of the presence of pages having a common parent in the sessions of different users.

2° The graph of the Web site.

3° The identity of the host page ([page 1] in the example) and the position of the link to the first of the pages in the aggregate ([page 1]→[page 2] in the example; the new link will be placed right before this link).

4° The content and the label of the new link, both obtained from the links to the pages composing the aggregate ([page 2, page 3, page 4, page 5] in the example).

In the current approach we only build aggregates composed of two pages. The aggregates can nevertheless be extended to more than two pages by several iterations of the algorithm.

2.2.2 Building a graph representation for the Web site

In order to be able to build the trees associated to user sessions, we need not only the logfiles but also a graph representation of the web site. Such a representation is built every time we want to process new logfiles, since short cuts and aggregates previously obtained modify the structure of the site. This representation can be huge for dynamic sites and this is why we are currently looking for alternative methods.

The nodes of the graph correspond to the (dynamically generated) pages and the (directed) edges to the links between two pages. The structure associated to each node contains the URL (or the entire query if the resource is generated dynamically, e.g. `/Webring/Consul.idc?Type=site&motcle=son&contenu=periph.htm`), pointers to the successors of the node and a set of flags.

The directed graph is built recursively, by following all the available links within every page. Obviously, no form is filled in, so we do not deal with interactive queries. When building the graph we make sure that any URL (query) corresponds to a single node. Note that this method can also be applied when some fields in the database contain source HTML files which include links (this is the case for the Validation site). Also, note that the graph of a site is usually not a tree.

2.2.3 Pre-processing the logfiles

An HTTP server stores all the queries served during a period of time in one or several logfiles, where every line corresponds to a single query, as in the following example:

```
194.2.149.85, -, 14/12/99, 18:50:03,…, /validation/default.htm, -,
194.2.149.85, -, 14/12/99, 18:50:05,…, /validation/accueil.htm, -,
194.2.149.85, -, 14/12/99, 18:50:05,…, /validation/somfr.htm, -,
194.2.149.85, -, 14/12/99, 18:50:08,…, /validation/images/li.gif, -,
194.2.149.85, -, 14/12/99, 18:50:08,…, /validation/images/gtra.gif, -,
194.2.149.85, -, 14/12/99, 18:50:09,…, /validation/images/facc.gif, -,
194.2.149.85, -, 14/12/99, 18:50:09,…, /validation/images/falg.gif, -,
194.2.149.85, -, 14/12/99, 18:50:10,…, /validation/images/fpro.gif, -,
194.2.149.85, -, 14/12/99, 18:50:25,…, /validation/requetes/alr1.idc, -,
194.2.149.85, -, 14/12/99, 18:50:52,…, /validation/requetes/pbr1.idc, -,
194.2.149.85, -, 14/12/99, 18:51:17,…, /validation/requetes/PbLien4.idc?PbTitre=sunspots,
```

We start by assembling all the logfiles corresponding to the period of time we are interested in, and then we filter out all the queries for files other than .htm, .html and .idc. Also, if the same HTTP server hosts several sites, we keep only the queries for resources of the site we want to modify.

2.2.4 Finding an adequate representation for user sessions

In paragraphs 2.2.1.1 and 2.2.1.2 we pointed out that in order to take the decision of building short cuts and aggregates we need to know the sequence of pages visited by each user, together with the corresponding time stamps. We found very convenient to represent each individual users session as a (set of) tree(s) – the chains of pages which are followed several times during the session are duplicated in the tree – and associate the corresponding time stamps to the nodes of the tree. In order to produce such representations we need the complete graph of the web site and the logfiles.

For any reasonably busy site, the queries sent by different users are usually mixed up in the logfiles. Since the decisions we make rely on the sequences of pages retrieved by individual users, we must be able to distinguish between individual users sessions.

2.2.4.1 Defining user sessions

A user is usually identified by his IP address. The difficulty comes from the fact that, over a long enough period of time, different users can share a same address: they share the same computer, or a DHCP server successively assigns the same address to their computers, or a router successively associates their addresses (internal to the organization) to a same routable IP address. We shall consider here that a session of one user is identified by an IP address, under the constraint that the period of time between two successive queries sent by the user is no longer than 30 minutes (1800 seconds). The value of this parameter (which is entered by the administrator when he launches our software) should be significantly reduced for common web sites.

Therefore, given the (pre-processed) logfiles, we start by grouping together the queries according to their IP address; then, for every IP address, we split the list of queries into individual sessions consistent with the time constraint mentioned above.

2.2.4.2 Building trees associated to user sessions

Having the list of queries corresponding to a users session and the graph of the web site (allowing us to identify the predecessors and successors of any node), we can now build the tree representation according to the following algorithm:

```
The current query is the first query in the list
The current tree is empty
As long as the list of remaining queries is not empty {
      Create a new node in the tree and label it with the current
      query, it becomes the current node
      If the current node is a successor of the previous node in the
      tree, link it to the previous node in the tree
      Else, backtrack until a node in the tree is a predecessor of
      the current node, and link it to the current node
      If no node in the current tree is a predecessor of the current
      node, create a new tree having the current node as its root
      Set previous node ← current node
      Set current query ← next query in the list
}
```

Note that, for every individual user session we can obtain several independent trees; indeed, the user may own several direct pointers to the resources of the web site (obtained e.g. with the help of a search engine).

Also, we lack precision when building the trees because of the presence of various levels of caching (single user, multi-user) and the use of the "back" button. Indeed, if the user is revisiting a page, the navigation tool may retrieve it from the cache instead of resending the HTTP request. Also, when the "back" button is employed by the user before switching to a new path, it may be difficult to identify the node from which the new path was started from several potential candidates; our choice here is to start the new branch as far from the root as possible.

When building the tree representations, we must also include with every node an estimate of the time spent by the user on the page corresponding to the node. These estimates are obtained from the time stamps present in the logfiles. If page A has page B as a son in the tree, then we can estimate the time spent by the user on page A by subtracting the time stamp of the HTTP request corresponding to page A from the time stamp of the request for page B. As an example, if the logfile contains

```
194.2.149.85, -, 14/12/99, 18:50:52,…, /pageA,
194.2.149.85, -, 14/12/99, 18:51:17,…, /pageB,
```

then we consider that the time spent by the user on page A is (18:51:17) – (18:50:52) = 25 seconds. Obviously, this time interval includes the delays introduced by the network, the HTTP/database server(s) and the client. Unfortunately, all these delays can vary within rather large bounds, depending on the location of the client with respect to the server and on the load of the network and/or the server(s). We consider that if one reads even a few lines of text on the page one receives, then the time really spent on the page is significantly higher than the sum of the delays.

There are three special cases deserving further explanation. First, when a page is the last one in a chain (that is, a leave in the tree), we decided not to take its occurrence into account. If the page ends the session of the user, we simply cannot compute the time spent by the user. If the page does not end the session, given the presence of (various levels of) caching, it is very difficult to know how much time did the user spend on this page and how much time he spent clicking the "back" button.

Let us now consider a page having several (i.e. more than one) sons in the tree. We decided to compute the time spent by the user on this page only once, using the time stamps

corresponding to the first branch. This obviously underestimates the time spent by the user on this page, but we didn't find any better solutions. Indeed, such a case occurs when some form of caching is involved (e.g. the user clicks on the "back" button, or he leaves this page on the screen and opens one of the successors in a different window) and we cannot obtain reliable estimates corresponding to the other branches.

As the last special case we consider pages which occur several times in a tree. The time spent by the user is computed for every occurrence, according to the context (the difference between the time stamp of the current successor and the time stamp of the page itself).

Under certain circumstances, however, the individual estimates we compute for the time spent by the user on a page can be entirely wrong: when the server is temporarily unavailable (a TCP connection cannot be established, so the user must try several times before the server can accept and answer to the HTTP request), or when the user sends several queries before receiving any answer (multi-activity of the user, due to long delays). Since we only use statistical measures based on these individual estimates (and never the individual estimates directly), we consider that the results obtained from such estimates are reliable enough if the logfiles contain many user sessions.

2.2.5 Statistical processing

Given the set of tree representations corresponding to the sessions of the users, some processing is required in order to identify the short cuts and aggregates which should improve navigation.

To evaluate the importance of individual pages for the users, we must compute the average time spent by the users on every page occurring in the logfiles. A page can be bypassed if it occurs frequently and the average time spent by the users on this page is very small, with a low standard deviation. Since we need reliable estimates for the average time spent, we only select as potential candidates the pages whose number of occurrences is reasonably high and for which the standard deviation is low. Statistical tests allows us to perform this selection: for candidates to getting bypassed, the average time spent is compared to the average time interval required for a user to get an answer to an HTTP request (15 seconds in our experiments). This value is either estimated *a priori* (as in our experiments), or by selecting the shortest time interval from each of the sessions and performing an average. For candidates to becoming the target of a bypass link, we compare the average time spent to the sum between the average time interval required for a user to get an answer to an HTTP request and the average time required for a user to read a few (10 in our experiments) lines of text. We are currently exploring alternative methods for making more reliable decisions. We eventually mark in the set of trees all the pages which should be bypassed.

Then, we must compute the joint frequency of the presence of pages having a common parent in the sessions of the users. We start by recording every pair of pages having a common parent in the set of trees, then we count the number of occurrences of every such set of three pages (parent plus two sons). The pairs whose frequency is beyond a threshold, set by the site manager, serve for building the aggregates (the two are aggregated). These pages are then checked for the "bypass" mark. Indeed, if one of the pages in a pair must be bypassed, then we should rather use the target of the short cut when building the aggregate.

2.2.6 Creating short cuts

To better understand how a short cut is created, let us consider again a chain …[page 1] $\xrightarrow{t1}$ [page 2] $\xrightarrow{t2}$ [page 3] $\xrightarrow{t3}$ [page 4] $\xrightarrow{t4}$ [page 5]… which appears very often, the average time spent on [page 1] and [page 4] (t1, t4) being high, while the average time spent on [page 2] and [page 3] (t2, t3) is very low. Then, if there is no direct link [page 1]→[page 4], we add to [page 1] (the host) a link to [page 4] (the target), right before the link to [page 2] (bypassed, together with [page 3]).

The label and the content of this new link is the same as the one of the link [page 3]→[page 4]. [page 1] is the only one that must be modified. Note that if [page 3] is dynamic, we cannot find the link to [page 4] directly (the link is usually not present in any of the files `page3.idc` and `page3.htx`); we must first obtain the actual [page 3] by following the chain from [page 1].

It is then rather easy to add a short cut if [page 1] is a static HTML page. Things get more involved when dealing with dynamic pages. Let us consider that every page in the previous chain is the result of an SQL query (present in a specific IDC file) formatted by a specific HTX file. How can we then add the link to [page 4] (page4.idc) on [page 1]? Which one of the files `page1.idc` and `page1.htx` do we modify? We can modify one of the queries on `page1.idc` to explicitly add the link to `page4.idc` to the text returned by the database (or create a new, fake query returning the link). Alternatively, we can choose to modify the template `page1.htx` and add the link to `page4.idc` at the right place there. We decided to implement the second solution which is more natural and can be readily applied to sites developed in PHP[1] or ASP.

To find the location where the link to [page 4] must be inserted (right before link to [page 2]), we must obtain the actual [page 1] by following the chain from the root of the tree and compare it to the file `page1.htx`. Note that in some cases the link to [page 4] cannot be inserted right before the original link to [page 2].

Let us now discuss the case where [page 1] is a frameset. A new link can only be included in a frameset if a new frame is defined. Given the difficulty of finding an appropriate layout for the new and existing frames in a general case, we decided to avoid adding new links (short cuts) within a frameset. If several pages should be bypassed between the frameset ([page 1]) and the target of the short cut ([page 4]), then we can add a link to the target on the first page in the chain which is not a frameset (e.g. [page 2]). This option was not implemented yet.

After creating a short cut, we have to modify the graph of the site (addition of the new link [page 1]→[page 4]), but also the trees representing user sessions. Indeed, the trees are further employed for building the aggregates, and graph of the site both for the aggregates and the addition of other short cuts after another data acquisition phase.

2.2.7 Creating aggregates

To better understand how an aggregate is created, let us consider that when on [page 1] many users retrieve [page 2] and [page 3], both of them being directly linked to [page 1]. If the frequency of occurrence of the pair <[page 2], [page 3]> is beyond the aggregation threshold, then we build an aggregate, denoted here by [page 2,3]. This aggregate is composed of the content of [page 2] followed by the content of [page 3]. A link to [page 2,3] is added on

[1] The PHP (or ASP) pages we are interested in can be seen as being composed of an HTX template and some code containing a request in SQL (as in an IDC file).

[page 1] right before the link to [page 2]. The label of this link is obtained by concatenating the labels of the links from [page 1] to [page 2] and [page 3]. While all this is rather simple from a conceptual point of view, we have to overcome several practical difficulties such as dealing with dynamic pages and/or with framesets, or building the link to the aggregate when the links to the parts are images.

According to the type of pages we wish to aggregate, we have three possible cases:

1° [page 2] and [page 3] are static pages: `page2.html` and `page3.html`. To obtain the aggregate (a new static file) we start by chaining the contents of the individual fields in the <head> section (such as the keywords in the content field of the tag <meta name="KeyWords" content="…">), including the <title>, then we chain the contents of the two <body> sections. Also, the name of the new aggregate [page 2,3] is a concatenation of the names of the two individual pages (`page2&page3.html`); the aggregate can be stored with the same path as the two parts (if they have the same path) or with a path which is specific for all the aggregates. The link to the aggregate is obtained from this path and the name of the aggregate.

2° [page 2] and [page 3] are dynamic pages (for every page we have an `.idc` file and an `.htx` file). With the current implementation, two dynamic pages can only be aggregated if they make use of the same database (the same `Datasource` in the two `.htx` files). This is a limitation of the IDC technology which can be avoided if the implementation employs PHP or ASP.

We have two possibilities here: build an aggregate which is either static (`.html` file) or dynamic (`.idc` file and `.htx` file). The first alternative is obviously the simplest, but we decided to avoid transforming dynamic content into static content and opted for the second alternative. We must then obtain a new `.idc` file and a new `.htx` file. The new `.htx` file is built according to the same procedure as the new `.html` file in the case of an aggregate composed of static pages. The aggregate `.htx` file is given a new, conventional name. Note that if the two pages make use of the same `.htx` file, then some of the information may be duplicated in the aggregate; a better solution in this case would be to duplicate only the fields between <%BeginDetail%> and <%EndDetail%> in the original `.htx` file in order to obtain the aggregate `.htx` file.

The new `.idc` file is obtained by keeping the `Datasource` field, entering the name of the new `.htx` file in the `Template` field, and listing the `SQLStatements` from the `.idc` file corresponding to [page 2] followed by the `SQLStatements` from the `.idc` file corresponding to [page 3]. The total number of `SQLStatements` must remain under 24 for the current implementation of IDC. A problem may occur if a same variable, which is present in the calls to the original `.idc` files, takes different values for the two calls. There are many possible solutions to this problem and they all impose various limitations, so we retained one of the simplest solutions: the variable is removed from the calls and its appropriate values are “hardwired” in the `SQLStatements` concerned. This aggregate `.idc` file is given a new, conventional name.

The link to the aggregate is obtained by chaining this path, the name of the new `.idc` file and the list of arguments from each of the two original links to [page 2] and [page 3] (with the conflicting variables removed).

The two new files (`.idc` and `.htx`) are stored under the same path, which is dedicated to the aggregates.

3° One of the two pages is static (e.g. [page 2]: `page2.html`) and the other one is dynamic ([page 3]: `page3.idc`, `page3.htx`). Again, we decided to avoid

transforming dynamic content into static content. We then have to build a new `.idc` file and a new `.htx` file. The new `.htx` file is built by including the content of `page2.html` into `page3.htx` and receives a new, conventional name. The new `.idc` file is a copy of `page3.idc`, but with the name of the new `.htx` file as the `Template`, and it is also given a new, conventional name. The two new files (`.idc` and `.htx`) are stored under the same path, which is dedicated to the aggregates. The link to the aggregate is obtained by chaining this path, the name of the new `.idc` file and the list of arguments copied from the original link to [page 3].

It is obviously very easy to add the link to an aggregate if [page 1] is a static page. When [page 1] is a dynamic page, we adopt the same solution we employed for the short cuts: we modify the template `page1.htx` and add the link to the aggregate `.idc` file at the right place there. To find the location where the link to the aggregate `.idc` file must be inserted (right before link to [page 2]), we must obtain the actual [page 1] by following the chain from the root of the tree and compare it to the file `page1.htx`.

We may encounter further difficulties in creating the link to the aggregate when the links to [page 2] and [page 3] appear on [page 1] as images or parts of images (maybe parts of a same image). The solution we retained is to create an anchor inside the aggregate at the beginning of the content of [page 3], to replace the link to [page 2] by the link to the (beginning of the) aggregate, and the link to [page 3] by a link to the anchor. It is possible to set up anchors inside `.htx` files and a link to such an anchor has the form given in this example: `file.idc?arg1=val1&arg2=val2#anchor`. However, if [page 1] is the result of an `.idc` request and the links are not present the corresponding `.htx` file, then these links are difficult to modify (they come from the database or are composed of elements from the database). In the current implementation we simply do not build the aggregate in such a case.

We do not aggregate [page 2] and [page 3] if any of [page 1], [page 2] or [page 3] is a frameset, mainly because of the difficulty of finding an appropriate layout for the aggregate and/or for [page 1].

After creating an aggregate, we have to modify the graph of the site by adding the aggregate and the corresponding link. We do not update the trees representing user sessions because the addition of an aggregate has no influence on the creation of new aggregates during the same session.

One can notice that we do not modify the layout of the individual pages composing an aggregate. Sometimes, this may the source of serious conflicts. However, we believe that there is no general solution to this problem.

2.3 Experimental results

The adaptive methods we put forward in the previous sections should be tested on web sites which respect the assumptions in § 2.1: they contain an important amount of fine-grained information bricks, they have a complex (and deep) link structure, and they are accessed by a large number of users. Since no such site was (entirely and locally) available, we only conducted experiments on two small sites, developed locally. The first one, called Validation site in the following, serves to perform comparisons between machine learning algorithms on several problems and is currently employed by a rather small community. The second site, called Webring in the following, puts together a number of presentations of computer peripherals, presentations which were developed by students in a distributed manner. The link structure of the two sites is rather simple and flat, but they both employ the IDC technology to

produce dynamic pages. The tests we performed on the two websites only allowed us to verify the algorithms and imagine possible extensions.

2.3.1 Results for the Validation site

Starting from a set of 34 logfiles (corresponding to 34 days), a short cut and an aggregate were created.

The short cut is set between the page corresponding to `/validation/requetes/pbr1.idc` and the page `/validation/formul/reframes.htm`. The first of these pages briefly presents the set of problems available on the site, while the last page allows any user to report new results obtained for one of these problems by filling in a form. The page which is bypassed here, `/validation/requetes/PbLien4.idc?PbTitre=sunspots`, is a brief report of the results obtained by various learning algorithm on the sunspot prediction problem. There are two remarks we should make here. First, the short cut does not fit well to the layout of `/validation/requetes/pbr1.idc`. Second, it is easy to see that the same short cut is useful for all the problems listed on `/validation/requetes/pbr1.idc` and not only for the sunspots problem; since the other problems are less well known, we do not find many enough such paths in the logfiles.

The link to the single aggregate generated is also set in `/validation/requetes/pbr1.htx`. This aggregate was obtained by assembling the pages corresponding to `/validation/requetes/pblien4.idc?PbTitre=Eau` and `/validation/requetes/pblien4.idc?PbTitre=Fisher`, which present in brief the results obtained by various algorithms on the water consumption (Eau) dataset and Fisher river dataset. The aggregate consists in two new files, `newidc.idc` and `newhtx.htx`.

2.3.2 Results for the Webring site

Starting from the set of logfiles obtained after two working sessions during which two groups of students consulted the site. We have deliberately limited the number of short cuts and aggregates to one.

The short cut is set between the pages `/webring/periph.htm` (a frame containing several taxonomies of the content) and `/webring/s27/accueil.htm` (the homepage of a site containing various information about digital cameras). The page corresponding to `/webring/consul.idc?type=site&motcle=image&contenu=periph` was bypassed.

The link to the aggregate was set on `/webring/s28/default.htm` (the homepage of a site presenting general information about computer networks). The aggregate was obtained by assembling the page corresponding to `/webring/s28/glossair.idc` (a lexicon of terminology related to computer networks) and the page `/webring/s28/contacts.htm`. Since one of these pages is dynamic, the aggregate also consists in two new files, `an .idc` file and `an .htx` file.

3 Directions in the development of adaptive hypermedia

We only explored here a particular case in a specific approach to adaptive hypermedia. There are many other research directions and in several of these directions a significant amount of work has already been done (see e.g. the Second Workshop on Adaptive Systems and User Modelling on the World Wide Web, http://wwwis.win.tue.nl/asum99/, or the International Conference on Adaptive Hypermedia and Adaptive Web-based Systems, http://ah2000.itc.it/).

Our approach here was to assemble a knowledge base out of fine-grained information bricks and let its global usage adaptively build up and modify a structure, starting from an initial design. We then focused on offline global changes applied to a knowledge base, changes from which every user (or most users) can benefit. But in many cases the adaptation to individual users or groups of users may prove very useful.

Also, further efforts are required in order to find out how can various adaptive mechanisms integrate smoothly with top-down or bottom-up methods for the development of hypermedia.

3.1 Adaptation to individual users or groups of users

We can first distinguish between offline (or inter-session) and online (or intra-session) adaptation. In offline adaptation, the input to the process of adaptation is the behaviour of the user (or of members of a same group of users) during the previous sessions. In online adaptation, the behaviour of a user during a given session has an influence on the continuation of the session itself. Building and progressively modifying user profiles is an important activity in both offline and online adaptation. The system acts according to rules which are developed most of the time offline, based upon the behaviour of the previous users. But if in offline adaptation the preconditions of the rules contain details about the identity of the user or a recognized profile, in online adaptation these preconditions must also take into account the perceived behaviour of the current user earlier in the session.

Obviously, a hybrid solution (offline plus online adaptation) can be employed.

A major difficulty here is to identify the user, or the group he belongs to. This identification can be explicit (the user identifies himself) or behaviour-based (recognition of a profile).

3.2 Integration with a top-down approach

In a top-down approach one usually starts by designing the structure of the hypermedia collection (e.g. knowledge base) to answer the specific needs identified. The content is developed next.

The new technology of symbolic links (s-links, [Verley and Rousselle, 2000]) supports a top-down approach, but can also help in the bottom-up development. S-links are typified links which satisfy a conceptual need expressed by the author and/or the user of a document. Also, while standard links are usually references to the location of a document, an s-link is a query to a link database which returns the reference to the document. This indirect addressing, together with a mechanism which automatically declares all the available links to the link database (an s-link is automatically declared and “comes to life” when it is activated for the first time), bring in most of the technical advantages s-links have over common links:

1° S-links support navigation by concept, i.e. navigation in a space which is different from document space and where both the nodes and the links have rather clear semantics. Implemented within a specific tool, navigation by concept should replace classical navigation, but can also assist or complement it.

2° S-links make automatic linking possible. However, problems related to synonymy and disambiguation have yet to be solved.

3° Even though an S-link is set as a link from a document A to a document B, it can also be seen as a 2-way link: the reverse link comes to life via the conceptual navigation tool, at the same time as the direct link (i.e. right after the direct link is activated for the first time).

4° S-links greatly simplify the entirely distributed development of content, by minimizing the exchange of information between developers. As a result of the definition of link servers, every developer or user can select one or several preferred link servers; some sort of certification method can also be employed. Again, some form of competition among content providers can be implemented through link servers.

In the s-link framework, adaptive mechanisms can be used for various purposes. Adaptation can act, to some extent, on the structure of the concept network to produce short cuts or aggregates. In the tool for navigation by concept, the items listed, their arrangement or number of items per level can adapt to the user or to groups of users. The choice of the document behind an s-link (the level of detail, the terminology employed) or the choice of alternative link server(s) can also be set to depend on the identity or the profile of the user. If link servers implement the competition between content providers, adaptive mechanisms could be employed for the selection of providers by a user or group.

3.3 Integration with a bottom-up approach

In a bottom-up approach, one attempts to build links between content elements out of more or less structured collections of flat multimedia documents, in a totally unsupervised manner or with a shallow description of the user needs in mind. S-links can be very helpful in this endeavour, both because of their conceptual advantages (typified links which satisfy conceptual needs) and because of their technical advantages (automatic linking, distributed development).

Learning and adaptation play a very important role here, and much of the existing work goes along this line.

3.4 Extending the adaptive behaviour to deal with other media

It is obvious that the techniques we presented for the adaptive structuring of information keep their attractiveness when extended to media other than text. However, the promise of adaptive hypermedia goes far beyond such straightforward extensions, and thinking of possible applications is probably the best way to evaluate these prospects. Here are a few suggestions:

Hypermedia global interfaces for the physically challenged, building upon the concept of economical interactions. Indeed, for the physically challenged even common interaction behaviours (many or most everyday behaviours belong to this class) may be tedious or even painful. By using adaptive hypermedia interfaces here, such behaviours need no longer be endlessly repeated.

Music explorer: multidimensional representation of music fragments, where the dimensions are either objective ones (such as rhythm) or more subjective ones. The choice and/or the construction of the subjective dimensions, as well as the selection of dimensions for a given user or group of users, can be performed adaptively. Also, one may be able to attain some form of mood control by defining trajectories in this (hyper)space. The definition of such trajectories can also be performed adaptively.

Hyper-video or tagged video (such an application already exists): parts of an image within a movie are links to other hypermedia and let a user get more related information, buy the item displayed, etc. In this case, the identity of the image parts which contain links, as well as the choice of the targets for these links, can adapt to the user or the context.

Adaptive video: in a synthetic movie or a videogame almost everything can be made adaptive: the general script and the details, the fillers for such variables as the characters, the scenery items, the sound, the (implicit) advertising, etc.

Such suggestions are only a glimpse to the endless possibilities of adaptive hypermedia (though, some of them broaden the scope of the concept).

References

Armstrong, R., Freitag, D., Joachims, T., Mitchell, T. (1995) WebWatcher: a Learning Apprentice for the World Wide Web, *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, March 1995.

Brusilovsky, P. (1996) Methods and techniques of adaptive hypermedia, *User Modeling and User Adapted Interaction* 6: 87-129.

De Bra, P., Calvi, L. (1998) AHA: a Generic Adaptive Hypermedia System, *Proceedings of the 2nd Workshop on Adaptive Hypertext and Hypermedia*, Pittsburgh, USA, June 20-24, 1998.

Perkowitz, M., Etzioni, O. (1998) Adaptive web sites: automatically synthesizing web pages, *Proceedings of IJCAI'98*.

Verley, G., Rousselle, J.-J. (2000) An evolved link-specification language for creating and sharing documents on the web, *Proceedings of CRIS 2000: Current Research Information Systems*, Helsinki, Finland, May 25-27, 2000.

Wexelblatt, A., Maes, P. (1997) Visualizing histories for Web browsing, *Proceedings of RIAO'97: Computer-Assisted Information Retrieval on the Internet*, Montréal, Québec, 1997.