Abstract

This paper describes a hypertext based system for teaching the C programming language. The course is presented using the Mosaic World Wide Web browser and differs from a conventional hypertext based course in that the material is individualised for each user.

When a user first begins the course they follow a standard path and are asked, using HTML-based forms, a series of questions about their background and knowledge of programming. The answers to these questions then direct the system to present subsequent questions to explore the relevant knowledge more deeply. After this phase the main C course is entered but the user’s view of the material is tailored according to the answers given in the initial phase.

During the course the user is given tasks to test their knowledge of the material. This in turn tailors the later material. For example, poor solutions to tasks mean that sections of the course are repeated possibly with different examples.

The system makes heavy use of the server’s ability to execute a program to generate a hypertext object. A user modelling system maintains information about the student.

This user modelling approach to customising hypertext for individuals has also been used by the authors in a movie information browser (the system recommends movies that it believes the user will like based on their user model) and an individualised news service (incoming news items are filtered based on the user model).

1. Introduction

Hypertext systems such as the World Wide Web hold great promise as a vehicle for delivering self paced instructional material. Like other hypertext systems, WWW enables the learner to select their path through the hyperspace and their own speed.

In this paper, we describe work that exploits other facilities of WWW to provide customised hyperspaces: these offer a variety of learning and presentation options without increasing the complexity of the learner’s hyperspace. This augments the choice of path through the space with the creation of different pages and links between them.
1.1 Diversity of presentation of the same information

There is considerable evidence that different people learn best in different ways. This means that one person who wants to learn some aspect of C will learn well when the information is represented in one way where another person would find that same information confusing and a barrier to their learning. This means that it is unlikely that a single presentation would be helpful for all learners.

Some of the differences between learners can easily be accommodated in current hypertext systems while others cannot.

For example, one important way to support different user’s needs is the commonplace use of marked glossary links which the user can select if they are unsure of the meaning of a term. However, there is normally only one glossary definition offered. The user may find its explanation unhelpful because it may rely upon the user understanding concepts they do not know. It may even use language that this user finds hard to understand. For example, it is commonplace for writers to use humour in documents to make them more pleasant to read. This is fine for users who understand the humour. For others, it can be quite puzzling.

It is also simple to provide additional examples, of various sorts, that illustrate an element of the language. These can be indicated by link that the user can select. So, for example, the user who likes a mathematical example could use the link offering that, while a user who prefers text-based examples would select the button for that. A problem with this is that the user who likes mathematical examples will, most likely, need to select that link every time they meet a concept. It would be more convenient for them if they could tell the system that they would always prefer to be presented with a mathematical example by default.

In addition to the above possibilities, there are many other forms of customised presentation that are desirable but are not easily supported with simple hypertext.

Perhaps the most significance difference between learners is the preknowledge that they bring to a new learning task. In the case of users learning C, the other programming languages they know make major differences to the way that they will learn the various elements of C.

They will be able easily to understand some aspects of C because these involve essentially concepts they already understand well. Even so, they will benefit from explanations that are cast in terms of the concept as they have met it before, with similarities identified and important differences highlighted. We could present such possibilities as additional links that, as in the case of the user who prefers mathematical examples, lead to text matching the major programming background groups. However, this would also add considerably to the choices the user needs to make, increasing the complexity of hyperspace navigation. In addition, the user would normally want to select the same class of such text since their background knowledge is largely stable during
their reading of the hyper-document.

In contrast to the aspects they will learn easily, there will be some aspects of C that they will find problematical precisely because of their background knowledge. In this case, the best presentation of material would carefully identify the bases for their likely misconceptions about C and provide information to help them develop the new concepts. It may also be helpful to give the user a good deal of consolidation of the new ideas, perhaps by taking a number of varied examples and working through several tasks that involve those concepts. All this may well be irrelevant and tedious for a user with a different background.

In the discussion of glossaries above, we touched upon another important area in which it is desirable to offer different presentations of the same material. This is the different ways of writing that suit different people. As noted above, humour is only effective if it is understood. In addition, if the writer’s sense of humour irritates the reader, it is distracting.

Another aspect of writing is its level of abstraction. Some users like very terse and extremely abstract presentations of new information. Indeed, programmers tend to like these rather more than non-programmers. However, others prefer a new concept to be presented in concrete terms with copious examples and careful explanation of each.

The common theme in the above is that we could use normal hypertext to offer the diversity that would enable a range of users to learn C from the same hypertext document. But the users would be faced with a greatly increased navigation problem, that would operate as counter to the benefits of the availability of explanations that they find more useful. Moreover, as they met each concept, they would be tediously selecting the same set of options, having to sort through the ones that they do not want to use.

We have already had extremely promising results in proving such customisation for Unix manual entries presented in a Stretch-text system (Lithgo 1991). A similar approach (Encarnacion & Boyle 1990) was also effective. Both these systems achieved the customisation by using a model of the user’s knowledge to customise the hypertext.

1.2 User models to customise hypertext

A user model is a collection of information about a user. Our architecture for customising hypertext documents is driven by a user model.

Before we describe that architecture in detail, we outline the role of the user model. This represented the user’s preknowledge and preferences.

In the hypertext learning environment for C, this means that we establish what programming languages the users, how well they know various concepts in those
languages as well as the user’s knowledge of relevant aspects of machine architecture. We also represent the user’s preferences for presentation of information about programming languages: abstract versus concrete; terse versus more detailed and explicit; active learning versus directed.

The tools for determining this and for storing the model are part of a toolkit for user modelling (Kay, 1990; 1994, Cook and Kay, 1994). Essentially, they use a range of sources to collect information. They also enable the user to inspect the user model and to contribute information to it.

We take the information in the user model and use it to customise the hypertext that we present to the user. So, for example, a user who knows Pascal well, likes terse, abstract explanations that use jargon as appropriate will navigate a hypertext that is in this form.

2. Implementation

The NCSA httpd server provides a facility known as the "Common Gateway Interface" for execution of a program in response to a user selecting an HREF or link. Parameters can be specified on the link and these are passed to the invoked program. The program is then expected to construct an HTML page with appropriate MIME headers and write it to standard output. The httpd server collects the HTML and returns it to the client browser. This facility was originally designed to allow easy construction of gateways to databases and other services.

The customisation system developed in this project is based on a set of programs that are invoked by the NCSA httpd server using its CGI facility. One program uses HTML forms to collect information from the user about their programming experience and learning style and another takes this information and tailors individual HTML pages of the course material according to their choices.

The overall architecture is shown in Figure 1.
2.1 User Modelling System

The user modelling system provides a representation for the information about each user as well as tools to manage this. The basic components of our user model are organised into contexts. So, for example, all the components for our model of the user’s knowledge of C are in the C-context. Within a context, a set of structural relationships is defined by a view.

Where we have little information about a user, we can use a stereotype view. So, for example, if we only know that a user regards themselves as a Pascal expert, we use the Pascal-expert stereotype to define their knowledge of the programming concepts relevant to learning C.

The user modelling toolkit provides various interfaces for querying the user. The initial interview is driven by the view defined for a simple form-based interview program that is invoked by the NCSA httpd server. This is ci, the concept inventory interface, which takes a view definition and uses this to define the order of questions given to the user to collect a quick assessment of their relevant knowledge.

Other tools are available for more sophisticated querying and for enabling the user to see their user model, along with its explanations of the components and justifications for the conclusions about the user. The toolkit also offers tools for reasoning about the user. These have to deal with problems of noise and non-monotonicity. The C meta-hypertext
makes use of a simple and efficient reasoning mechanism based on endorsements.

The user modelling system is currently implemented as a set of library functions. We plan to make this a user model server with a simple access protocol.

2.2 Authentication
In order to maintain a user model and use it to customise HTML pages, the current user must be identifiable and there must be some form of authentication. Our system uses the simple authentication system provided by NCSA httpd that requires a user to enter a user name and passwd before access to certain directories is allowed. Since all pages in the hypertext C course are constructed using programs from the ‘cgi-bin’ directory it is a simple matter to require a valid, authenticated user before access to the programs is granted.

The current username is made available to the programs in the environment variable REMOTE_USER. This name is then used to consult the user modelling system for information about the user.

Before a user accesses the course for the first time, an account and passwd must be established as well as an initial user model or stereotype.

2.3 Concept Inventory
The Ci program consults the user model using a particular view and context given as a parameter on the HTML HREF or anchor. In the case of the C language course this is simply "c" but the system has been used for other applications such as system for giving advice on movies and a system to construct a personalised radio programme. In those cases, the view contains information about the user in that context.

Included in the view are questions that can be presented to the user to elicit the necessary information. Ci constructs an HTML form with these questions and returns this to the httpd server.

When the user has filled in the form and clicked on the Finished button, the answers are returned using the HTTP POST method. The Ci program analyses the answers and updates the user model appropriately. Only a modest number of questions are asked and the rest of the user model is derived from stereotypes.

2.4 Form of Course Pages
The course component pages are written in HTML but are augmented with a customisation language that describes how it should be customised for the current user. The PT program is configured to use another program to process the customisation language, so any suitable language could be used.
For the C course we use the C preprocessor (cpp) since it is readily available and powerful enough for this application.

### 2.5 Customisation

Apart from the process of gathering initial information about the user, all access to course pages is through the ‘PT’ program.

PT is run by httpd and is passed the name of a page that is to presented to the user. The user model is then consulted using the ‘c’ context for the current user (taken from the environment variable REMOTE_USER) and the resulting information turned into a sequence of ‘#define’ lines. PT then invokes the customisation language processor (cpp in our case) and pipes into it the sequence of #define lines followed by the raw course page. The output of cpp is fed back to httpd as the HTML page to be returned to the user’s WWW browser.

Here is a short example showing a fragment of a course page. The user has indicated that they are an expert Pascal programmer, have no knowledge of Basic and prefer an ‘active learning’ style of instruction.

The user clicks on the following anchor:

```
<A HREF=http:pt?C1/C1.2>
```

The raw course page includes the following:

```
<P>
A C program is a collection of functions.
One must be called <TT>main</TT> and this runs first.
#if PASCAL > 0 more than minimal Pascal knowledge?
As a Pascal programmer you should note that the line
<TT>main()</TT> does <I>not</I> have a semicolon at the end of it.
#endif
```

PT adds a MIME (RFC1521) header and, after consulting the user model, the following lines:

```
#define PASCAL 3
#define BASIC 0
#define Active-learner 1
```

This is then passed to the C preprocessor and the final text is:
AC program is a collection of functions. One must be called \texttt{main} and this runs first. As a Pascal programmer you should note that the line \texttt{main()} does \textit{not} have a semicolon at the end of it.

As well as modifying the text shown to the user this customisation technique can be used to determine which path through the course is presented. This is done by presenting a link to one page in one situation and a different link for another.

The technique can also be used to vary the appearance of pages and make it easier to move pages to other machines.

For example, at the end of most pages the user is presented with a \textit{next} link and a right arrow icon. This may be represented as:

\begin{verbatim}
<HR>
#if (Active_learner > 0)&&(PASCAL > 0)
  <A HREF=http://pt?C1/C1.1.1>
#else
  <A HREF=http://pt?C1/C1.2>
#endif
</A>

This indicates that if the user is an ‘active learner’ and has some Pascal experience they should move to page C1.1.1 when they select \textit{next}, otherwise they move to C1.2.

The source for the right and up arrow icon data is also represented using a \texttt{#define}ed symbol. This makes it easy to change the form and location of the arrow data.

\section{Discussion and Conclusions}

Essential elements of a hypertext are the pages, links between them and the semantic structure that exists between the pages. A \textit{metahypertext} allows all of these to be dynamic. Our architecture uses WWW to deliver a metahypertext. It has three essential elements.
The first is the user model and the interaction that helps define it for each user. We have implemented a WWW-based interface to do concept inventories that seed the user model.

The second aspect is the use of interaction with the user as a means of traversing the hypertext. This makes the interaction far more active than the more usual reading of a hypertext.

The third critical element of the architecture is the use of executable CGI facility. This permits the generation of arbitrarily customised hypertext pages.

4. References

R Cook, J Kay (1993), ‘Tools for viewing um user models’, SSRG Report 93/3/50.1, Dept of Computer Science, University of Sydney, Australia


J Kay, ‘The um toolkit for reusable, long term user models’, SSRG Report 94/3/36.2,


Bob Kummerfeld (bob@cs.su.oz.au) is an Associate Professor at the Basser Department of Computer Science, University of Sydney.

His research has been in the area of message handling systems and operating systems. Bob was co-developer of the software that underlies the ACSnet message network in Australia.

Judy Kay (judy@cs.su.oz.au) is Senior Lecturer at the Basser Department of Computer Science, University of Sydney.

Her research focuses on user modelling, especially for intelligent teaching systems and information filtering. She has developed a toolkit for user modelling with particular support for a range of acquisition methods. These are being used in a long-term research project involving modelling and coaching of a large number of users.

Judy co-chaired the Workshop on Agent Modelling for Intelligent Interaction at the 1991 International Joint Conference on Artificial Intelligence. She is a member of the editorial board for the journal ‘User Modeling and User-Adapted Interaction’.

Author contact email address: bob@cs.su.oz.au (Bob Kummerfeld)