EPCC-SS99

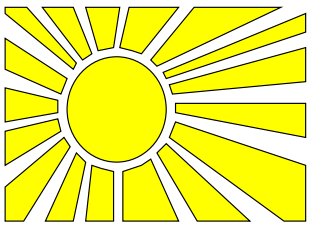
Knowledge Trees - Web Annotation Scheme

Alexander Presber

Supervisor: Dr. Mario Antonioletti

Abstract

The idea of community shared webpages is a logical extension to the static one-way-interactivity of normal webpages. To have the possibility of making remarks or giving extra information the author did not include in the original page can potentially increase the value for all users accessing. A basic annotation system for the WWW was implemented that allows users to comment on a page or add HTML content to it. We explore the possibilities of using dHTML and dynamically written HTML to provide a quick and easy-to-use interface on the client side, avoiding server access where not necessary.



Contents

1 Introduction

Many names are used to describe projects similar to ours, namely

- conferencing systems,
- online communities,
- shared webpages,
- annotation systems,

Their common aim is to allow additions to be made to existing resources on the World Wide Web by people who do not have direct control over the pages concerned. The approaches taken to allow some form of annotation differ in a lot of aspects:

- storage point of the annotations,
- allowed depth of nested annotation,
- resolution of user access permissions (groups, levels of writing permissions),
- supported browsers or additional tools needed to make/view the annotations,
- ease and speed of the interface.

It seems interesting to explore the possibilities of modern web browsers like NN 4+ and IE 4+ [1], namely dHTML [2] and server side CGI [3] techniques to provide a fast and convenient annotation system interface.

Lets first look at a “normal” server access.

1.1 Go surfing

As shown in Fig.1 a browser sends an HTTP-Request, including the URL of the required page, to the server which then sends back the document requested. Requests *can* include data, e.g. form data in GET or POST requests, but they always result in a document being sent to the browser.

Surfing the web you might end up finding a page like the cryptic buffalo page (Fig.2).

Then you probably feel lucky, if you have an annotation system at hand.

1.2 Comparision of existing systems

Two typical web annotating systems have been chosen, namely the *Xerox sparrow community* [1] and *Thirdvoice* [2] illustrating the following two ways of adding information to a web page:

[1] Netscape Navigator and Microsoft Internet Explorer version 4 or higher.

[2] *Dynamic Hypertext Markup Language*, refering to the capability of a web browser to dynamically change properties like position, visibility and style of elements of the web page.

[3] *Common Gateway Interface*, specification for script generated pages such as results of database queries.

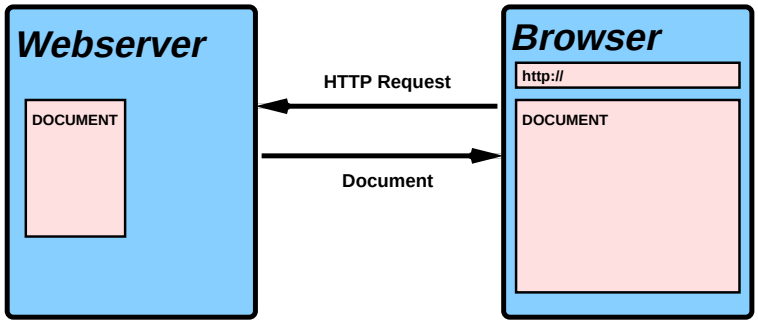


Figure 1: Simple HTTP-request.

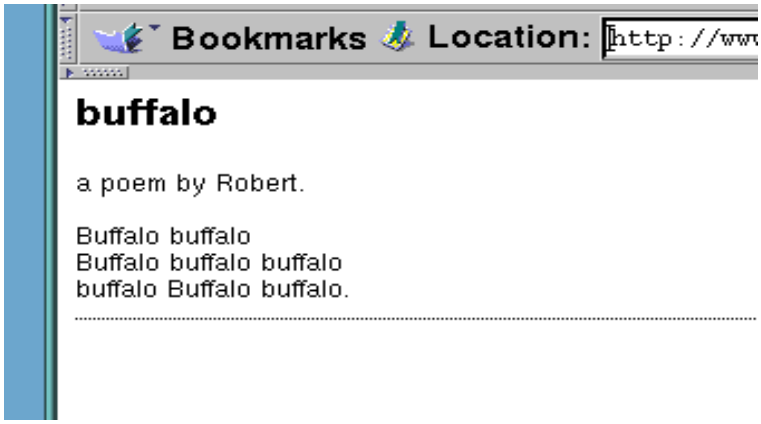


Figure 2: The cryptic buffalo page.

- *annotations managed by the webserver, using a* **CGI-interface**
  This resembles writing a letter to the editor of a book, saying "Consider changing this chapter, because, as I found out ..." though in this case we deal with a robot editor. A page update is necessary for every visual change on the page (like displaying a form to input your annotation) which results in increased server traffic.
- *third party annotation system, using a* **plugin**
  This is more like writing your remarks into the book you borrowed from the library.
  The author of the page has no control over your scribblings, which is of course a two-sided sword. This approach needs browser add-ons but results in much less traffic, because the server gets involved only, when data changes.

These are discussed below.

1.2.1 Sparrow community

Sparrow community is an "old style" approach that works on older browsers as well. It principally accepts HTML annotations, although the implementation filters the user input and accepts links only at certain predefined places. Fig.3–5 illustrate a typical interaction.

Fig.6 shows a sample page containing a "hot list".

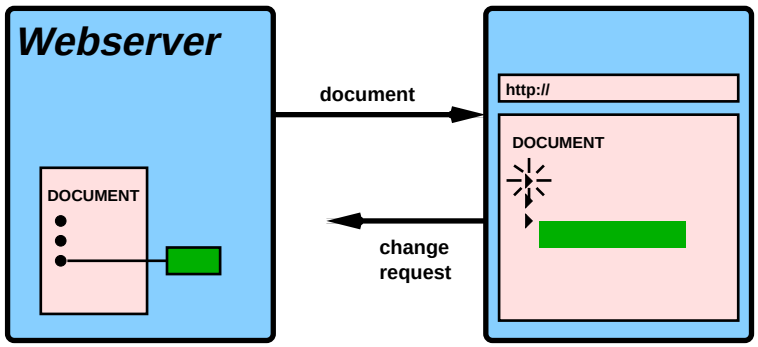


Figure 3: The document sent to the browser contains hot spots (triangles) which indicate, that an annotation can be added at this location. If the user clicks on one of these, a *change request* will be sent to the server.

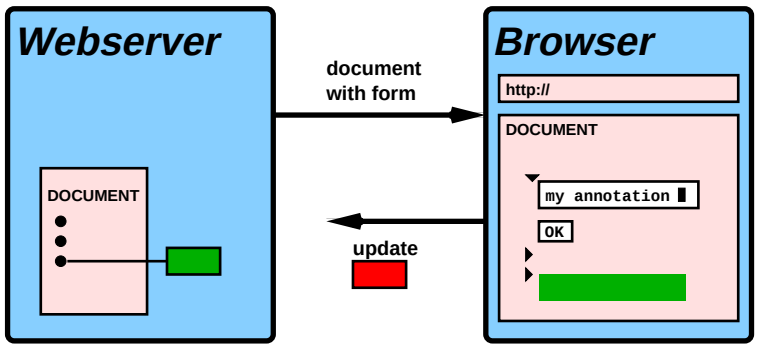


Figure 4: The server sends the document, containing an input form at the appropriate place. When the user presses the submit button, an update request, containing the input text is sent to the server (more precisely to an update script on the server).

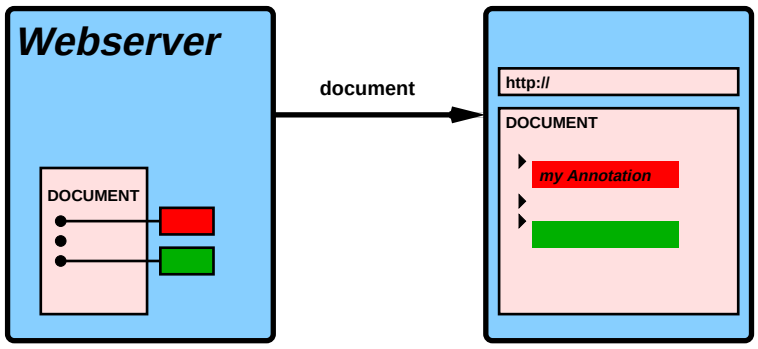


Figure 5: The updated document gets sent back to the browser as result of the request.

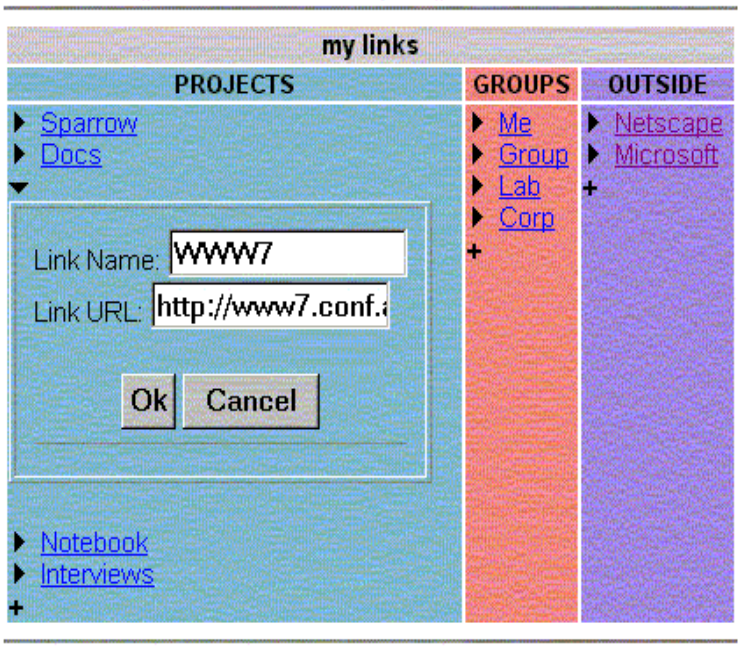


Figure 6: Sparrow community screenshot.

1.2.2 Thirdvoice

Thirdvoice is a system that consists of a plugin and an annotation server. Webpage annotations are stored on this server along with the address of the referred page.
Thirdvoice accepts only ascii text. The simplified dataflow for this system is explained in Fig.7 and 8.

1.3 The perfect annotation system

What is meant by *perfect annotation system* is one that takes advantage of all the above.

- The system must work with “normal” browsers, that is version 4 or above of Netscape Navigator (NN) and Internet Explorer (IE) without *plugins*. Support for version 3 browsers and LynX would be better, but are limit our possibilities for dynamic display.
- A further requirement would be to have robust, human-readable files, easily extensible to XML or database links like those used in IE5.
- For the time being the system must work with existing pages that rely on, possibly bad-formatted, HTML 3.2.
- Support for user authentification must be provided to protect other people’s annotations while making changes on our own annotations possible.
- Server access must be cut down to a minimum, that is for loading the data and finally making the changes. All actions in between should be taken care of at the client end.

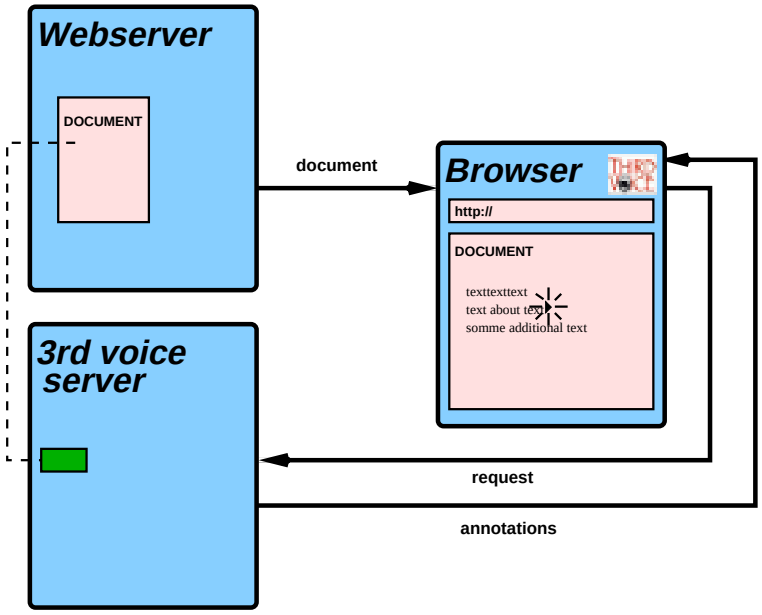


Figure 7: If thirdvoice is switched on, the browser plugin checks on the thirdvoice server, whether there are annotations on the current page available and, if so, downloads them.

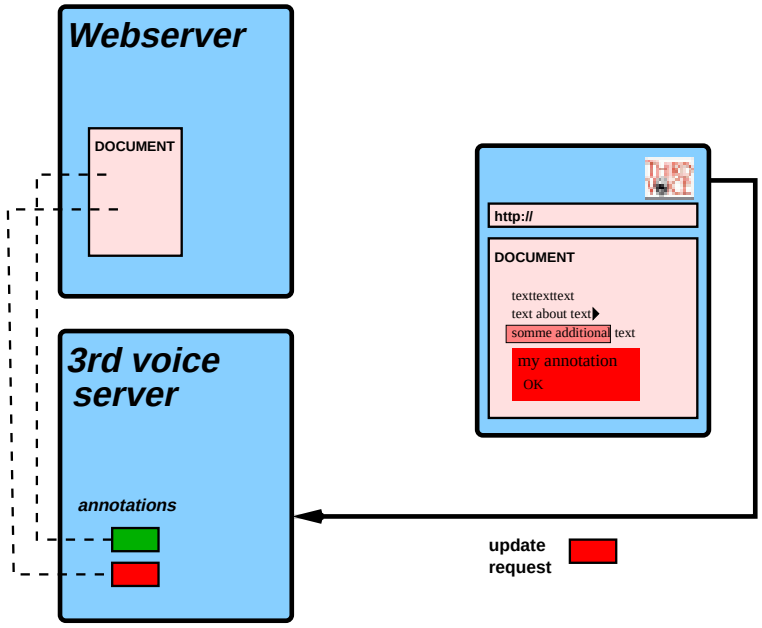


Figure 8: The input of new annotations uses a dHTML layer interface, the finished text is stored on the thirdvoice server.

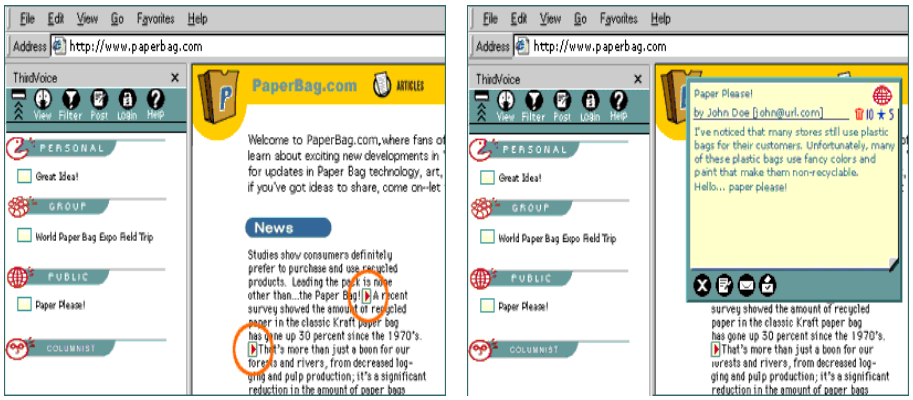


Figure 9: Thirdvoice screenshot; the circles point out, where the annotations are.

- Annotations should be possible at (almost) arbitrary locations.

Two decisions had to be made in order to implement a working prototype, even if the advantages may not be instantly obvious.

- The annotations are stored in the original file, *in situ*. This was adopted mainly for simplicity and avoids the necessity of handling multiple files.
- The annotations shall be displayed *inline* too, that rules out extra windows and frames to display them. (The use of windows makes the system slower, its more easy to loose the total overview, especially of the treelike dependance). A split window with two frames would be a common way to display a document and additional nonstatic information, but this is only practical if the depth of annotations is limited to one, which is very restricting. Also frames have a bad reputation when it comes to bookmarking and printing the page.

2 Principles

2.1 Storage

The storage of annotations, especially of the position of an annotation is not as straight forward as it may seem at first glance. Defining the position by simply counting words or sentences does not work if the content of the page can be changed after the annotation was put in. The word that was annotated might even disappear during such a change.
Two ways to tackle the problem are given below:

- String position trees [4], used in ComMentor [5]
  String position trees are based on position identifier strings. As [5] points out, this system is robust against modifications of the underlying document. When using this system, each annotation is a separate file. If its position in the annotated file before a change can *not* be determined afterwards, the annotation slides to the end of the parent file.
- physically embedding the annotations, using an SGML-like nested structure
  This is much less convenient if no redundant information is stored along with the annotation. Still there are ways to keep the annotations at their original place, e.g. using a tool like the UNIX `diff` programm.

The first approach did not seem realisable in the amount of time given, though principally more favourable. In the existent program version the annotation are embedded.

2.2 Representation of hierarchy

The annotation hierarchies are reasonably represented as a tree structure. In the chosen approach the path names are lists of numbers. A piece of text can be addressed by its parents path plus a distinct number. Annotation branches have a path name as well. Fig.10 shows an example.

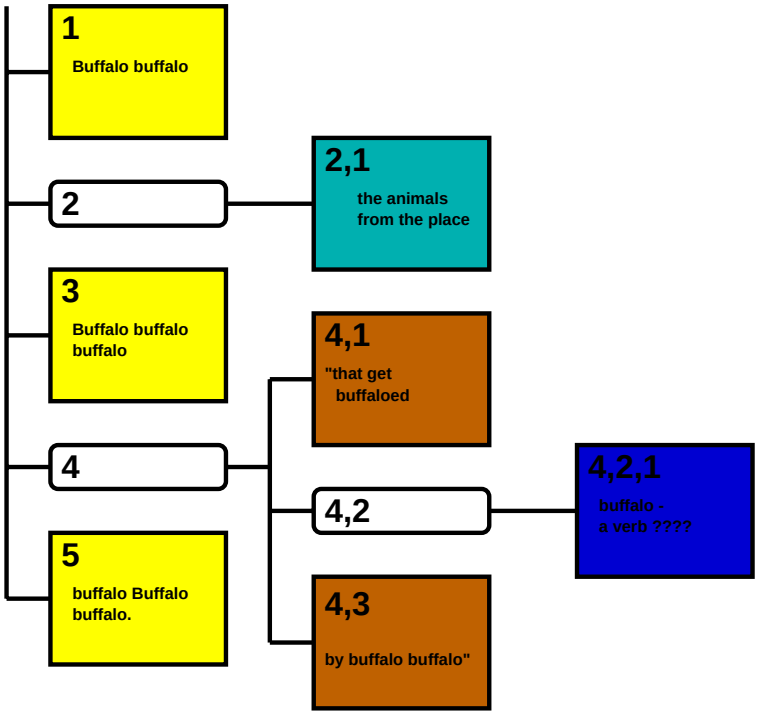


Figure 10: Knowledge tree for the mystic buffalo page (Fig.2).

2.3 The server

The Web server hosts the web pages with the embedded annotations as well as the Perl scripts to work on them. Whenever a browser requests a document, a Perl function parses the document and creates a tree structure for further processing. Another function sends an HTML frameset with a static JavaScript library and dynamic JavaScript arrays, containing the parts of the page, to the browser. Fig.11 illustrates this.

2.4 Client side interface

When using our system, the browser does not call the document directly e.g. `http://www.anno2000.com/page.h`
but calls the page through server side script e.g. `http://www.anno2000.com/cgi-bin/show.pl?file=pag`

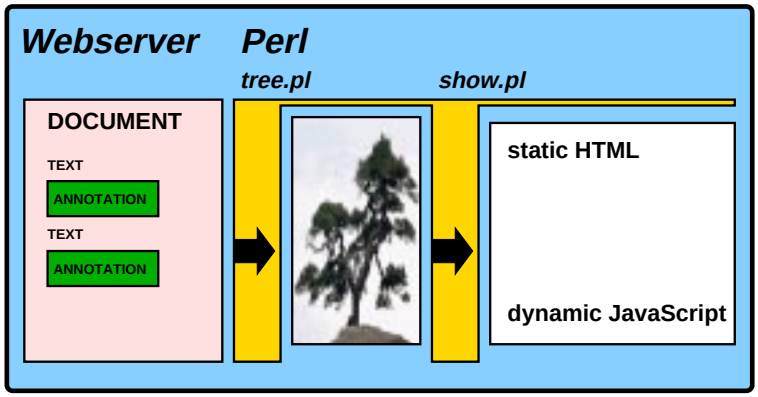


Figure 11: Structure of the Anno2000 server.

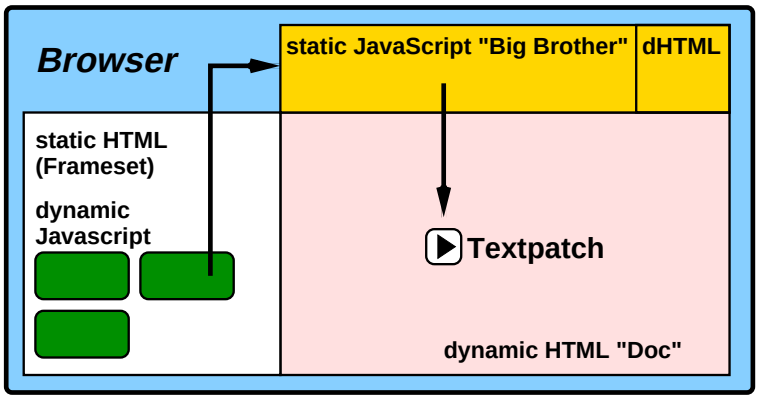


Figure 12: Structure of the Anno2000 client.

What gets sent back to the browser is like a DIY kit to display the page. The kit contains a frameset with two frames: the “BB” frame, that contains only HTML for two buttons and a status information layer, as well as a JavaScript library to dynamically create different HTML pages in the other frame, called “DOC”. The necessary data is in the header part of the frameset. As long as the URL does not change, the browser takes care of showing and altering elements of the page.

2.5 Permissions and access

There are different means of user access restrictions. We use the NCSA server side authentification mechanism using a directory specific `.htacces` file. This means that new users will have to apply for an account and a password. The complication of having to explicitly log into the server is outweighted by the benefit of higher security and control by the Webmaster. This is to ensure that users trying to sabotage an existing document can be removed easily from the `passwd` file. Users can edit and delete and annotate their own files but can only annotate other peoples text.

The script runs with a special user *id*, a user that does not belong to any group and owns no files apart from the annotateable documents. The path of a filename passed to one of the scripts is

automatically replaced by the `$Documentroot` given in the file `globals.ph` to ensure that users cannot escape out of the HTML tree.

3 Design details

3.1 File format

On the server side a couple of Perl scripts works on the files containing the Web pages and the annotations. These files are principally pure HTML. HTML comments are used as pseudo tags, to mark the beginning and end of annotations, namely:

begin	`<!-- ANN AUTH="byrner" TIME="1999" -->`
end	`<!-- END ANN ->`

For the mystic buffalo example, this might result in a file like this once it has been annotated:

```
<HTML><HEAD></HEAD><BODY>
Buffalo buffalo

<!-- ANN AUTH=''byrner'' TIME=''1999'' -->
The animals from the state.<!-- END ANN --><BR>
Buffalo buffalo buffalo

<!-- ANN AUTH=''byrner'' TIME=''1999'' -->
''that get buffaloed

<!-- ANN AUTH=''presber'' TIME+''1999'' -->
buffalo - a verb ?<!-- END ANN --><BR>
by Buffalo buffalo''<!-- END ANN --><BR>
buffalo Buffalo buffalo.
</BODY></HTML>
```

3.2 Perl datastructures

The most important Perl datastructure is the *knowledge tree*, this is represented by a tree structure of the form:

`@child : ($text,$child_array_ref,$path_reference,$author,$time)`

where the reference $child_array_ref points to an array of @child structures. The `child`s contain *either* text *or* references to childs. So a `child` with HTML content can be recognized from its `$children_ref` being empty.

3.3 JavaScript datastructures

JavaScript (JS) gets a *flattened* knowledge tree in a way that is suited to Perl→JavaScript communication, namely a Perl created JS array used to construct JS objects, called "patches" from

now on, when referring to the JavaScript objects. An element of this array has the following subelements, each with an example given:

	text	author	time	path	(indent)level	permission
c[0]=(	'buffalo',	'byrner',	'Sun Sep 5 19:42:07 1999',	'1,1',	1,	1);

The member variables and functions of the patch object are shown in Tab. 1.

3.4 The display engine

The screen updates are being taken care of by a JS function called `redraw()`. It is a loop over all patches, showing, if `patch.level<BB.showLevel`:

flag	shows
outVisible + button	menu button
ownVisible	text

3.5 Parts

The code for the subsequent scripts can be found in the appendix.

3.5.1 `tree.pl`

This perl module parses the document-string and returns a tree like described in 3.2. All other scripts call `tree.pl` initially to obtain this abstract representation of the document.

3.5.2 `show.pl`

The `show.pl` script writes the frameset document to the browser. In the <HEAD> of the frameset, all information relevant to JavaScript is stored like:

nrChildren	number of patches
File	current HTML-source
procEditScript	name of the edit script
procAddScript	name of the add script
killScript	name of the kill script
Anns	array of valid annotation point tags
state	state of the script :
	`(current annotation, showLevel`
c	array of patches
	`(text,author,time,path,permission,(display) level,`
	`end of this branch,subpatches)`

Also there is a function called "framedoctor" that gets called from the the BODY-onload-function of the DOC frame and returns true for the first time called. When the DOC frame gets restored and the top frameset isn't (this happens when using the browser BACK button), the function forces the frameset to be reloaded.

this.id	integer	index for self reference
this.author	text	author of annotation
this.time	text	time
this.text	text	actual content
this.path	text	path (for calling the scripts)
this.level	integer	annotation level (for indent)
this.permission	boolean	permission (0≡r, 1≡rw)
this.offend	integer	last patch that is part of the branch
this.patches	integer	annotated text consists of >0 patches
this.button	boolean	patch has button
this.outVisible	boolean	visibility of parent
this.ownVisible	boolean	own visibility
this.state	text	(‘ ‘|‘annotate‘|‘edit‘|‘annPoints‘)
this.last	boolean	1 if this is the last patch of an annotation
this.par	integer	current paragraph (to be annotated)
this.edit	function	
this.eCancel	function	cancels edit mode
this.annotate	function	
this.aCancel	function	cancels annotate mode
this.kill	function	kill annotation
this.hide	function	hides annotation and children patches
this.show	function	shows annotation and children patches with ownVisible==1
this.drawBt	function	writes the HTML for the approbriate menu button
this.info	function	writes the info for mouseOver menu
this.editHTML	function	writes the HTML for the edit form
this.showAnnPoints	function	writes the HTML for annotation points
this.sapCancel	function	cancels the showAnn mode
this.annHTML	function	writes the HTML for the add annotation form
this.textArea	function	writes an HTML textarea with the appropriate hidden inputs

Table 1: Elements of the patch object.

3.5.3 proc_add.pl

This script adds new annotations into a script.

I/O	from/to	name	content
I	query	*file*	file being worked on
	query	*path*	path of the text “child”, the annotation takes place in
	query	*par*	paragraph of the annotation, tagnumber
O	file		tree with new annotation
	browser		`"Location:show.pl?file=[file]&state=[state]\n\n";`

3.5.4 proc_edit.pl

This script is responsible for editing as well as deleting annotations (in which case it simply writes *nothing* at the place of the annotation).

I/O	from/to	name	content
I	query	*file*	file working on
	query	*path*	path of the child, that is to be changed
	query	*action*	(edit|kill)
O	file		annotation tree with changed annotation
	browser		`"Location:show.pl?file=[file]&state=[state]\n\n";`

3.5.5 global.ph

This file contains (desirably) all constants, most importantly those which have to be customisable. The most important are listed in Table 2

constant	meaning
$Docroot	allowed document directory
$Defaultdoc	document to be processed when script gets called withot file argument
$Cgi_bin	Perl script home
$Wwwroot	URL of documents
$ANN	Perl pattern for annotation start
$ANNOUT	Perl output -”-
$ENDANN	Perl pattern for annotation end
$ENDANNOUT	Perl output -”-
$Anns	allowed annotation points
$JSAnns	-”- converted to JS list’

Table 2: Constants from `global.ph`.

4 Example

In Fig.13 - 15 a working example of Anno2000 is shown – humbly presenting the to do list.

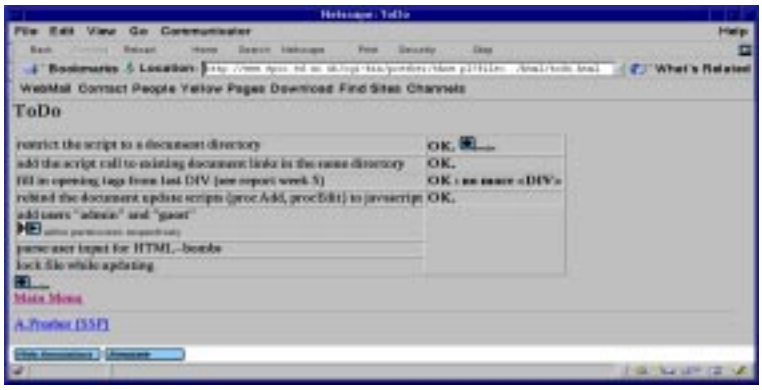


Figure 13: An example annotated table with three annotations, the middle one is opened, the others still show the author and, when moving the mouse over the box, the first words and the date of creation of the annotation.

Figure 14: The menu that has appeared at the middle annotation is the menu type for annotations that the user can edit. The entries mean: close menu, hide item, edit item, kill item, annotate.

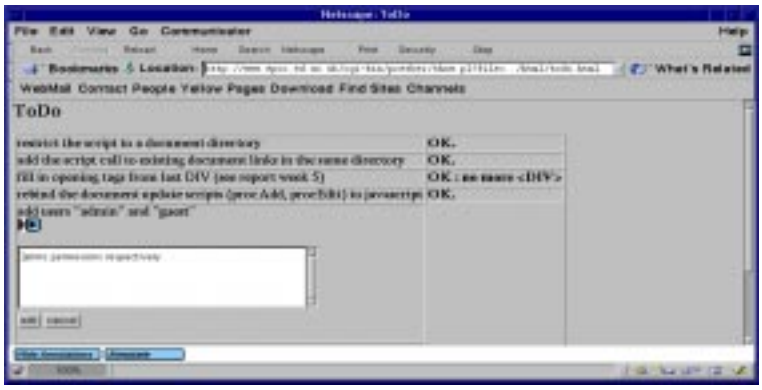


Figure 15: Inline edit field.

5 Shortcomings and design problems

- The storage format is very rough and it should be thought about using several files and a database, either implementing some sort of referencing mechanism basing on redundant data and so describing the annotation point or including just a reference to the annotation file(like `<!- ANN FILE="ann0001" ->`).
- A file locking mechanism is urgently needed. It would probably be a good idea to implement an additional pseudo tag for that reason, which contains the time of creation and becomes invalid after a certain time, thus avoiding zombie locks.

6 Tales from the WildWildWeb

6.1 New lands

First, as shown in Fig.16, all changes to the display except for scrolling were made by calling CGI-scripts, namely `show.pl`, `add.pl` and `edit.pl` depending on the required interface elements on the page. The actual changes in the document were made by the two `proc_` scripts, which, when finished, changed the browsers location to the `show.pl` script again.

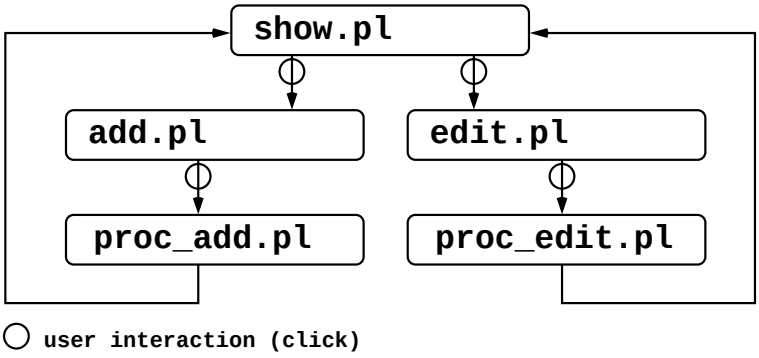


Figure 16: The first approach.

This worked very similar to the *Sparrow* system described in chapter 1.2.1. Everything was nice and slow.

6.2 Why dHTML ?

Why ask the server for a piece of HTML, when nothing really has changed on the serverside ? Why not send everything to the browser and shift the pieces around with JavaScript (see Fig.17)? Parts of the text can be easily hidden or replaced by a <TEXTAREA>, as long as everything gets sent to the browser at first. The IE4 document object model makes the actual HTML accessible and rewritable via JavaScript. The Page gets dynamically rerendered after the change. *That* is the way to do it but as long as NN does not support it, everything has to be put into layers and hidden by hand. But if that is done, the advantages are *very* short response times and a flexible layout. Is this not what dHTML is for ?

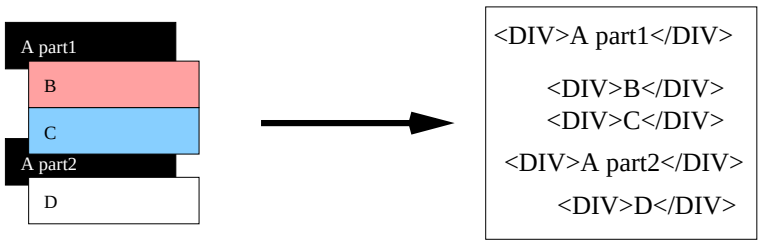


Figure 17: The dHTML <DIV> constructs required to build a page with hierarchical levels of visibility.

6.3 Why not ?

Without changing any of the tree parsing routines the display functions were completely rewritten. The single parts of the document that one must be able to individually hide or show were put into <DIV> ... </DIV> brackets. All the positioning was done by a JavaScript function after loading. The `display_all()` routine is as compact as this:

```
   function display_all(){ // "packs" all visible DIVS
    var position = yOffset;
    for(i=0 ; i<nrChildren ; i++){
child[i].move(xOffset + indent*level[i] , position);
button[i].move(xOffset + indent*level[i] - buttonSize , position);
if (visible[i]==2){
    position = position + child[i].getHeight();
    child[i].show();
    button[i].show();}
if (visible[i]==1){
    position += buttonSize;
    child[i].hide();
    button[i].show();}
if (visible[i]==0){
    child[i].hide();
    button[i].hide();}}
    return 1;
}
```

The result was an extremely fast and dynamic page. Unfortunately this also led to problems, the reasons of which are (in increasing order of severity):

NN does not extend the document dimensions when elements are placed outside of the current borders. The actual size of the different items can only be determined *after their rendering* by the browser. This forces us to provide a big enough *empty* table, that rerserves the space but might be too big.

An annotation branch could begin *within* an HTML structure such as a list. That means that we encounter *overlapping HTML sections* (only in the HTML that is sent to the browser, not the data file of course) like:

```
<DIV>
  <UL>
    <LI>something
</DIV>
<DIV>
  some annotation
</DIV>
<DIV>
    <LI>something completely different
    <LI>another point
  </UL>
...
```

What does the browser do here ? It silently closes the <UL> environment (the same for <P>, <TABLE>) when seeing the </DIV> tag. Later, of course, it stumbles over the following <LI> tags and stops parsing. This is a principal problem of HTML, more precisely of the structure of SGML – *No overlapping.* Even if it is only for the sake of *visibility*.

There are two approaches for a workaround at hand:

- put an opening <UL> after the end of the comment, thus restoring the environment before the intermediate <DIV>. This causes some extra complexity, because the information about the environment in which the annotation has been made must be stored in the tree. What about tables ? Reopening them would not work in most cases anyway, namely when COLSPANs are used.
- do it another way.

Annotating tables and lists is important enough to skip this approach. Back to the beginning ?

6.4 The last stand

What about a mixed approach ? JavaScript creates the pages on the fly but with included dHTML menues right at the point of annotations. This gets a bit tricky. Writing to the document that is just beeing displayed via `document.write()` causes the document to be deleted – including the JavaScript just beeing run. Not our intention. First a self-reproducing JavaScript was written, that sends itself *in one piece* to the browser, just changing an internal state. This is a theoretical informatics problem called a “quine” [3] and with a little help from some usenet groups it finally worked, but the result is rather unreadable and difficult to extend.

There is a much more reliable way: an invisible frame, that contains the script and data and a dynamic document frame that just uses 100% of the window. This is illustrated in Fig.18.

As a welcome side effect one can even create the <TEXTAREAS> and all the bureaucratic overhead on the fly, all that is needed are the original pieces of HTML, stored in a structure that represents the hierarchy of the document. Everything seems seems to be o.k. like that. First we tried using “floating” <DIV>s (position:relative) as menu anchors but a bug in NN 4.5 [4] forced us to use the actual menu *images* as position anchors instead, which works well. Finally it was decided not to neglect dHTML totally. The final display engine consists only of a Perl script

[4] It does not always build a complete document tree, when encountering <DIV> or <SPAN> in tables.

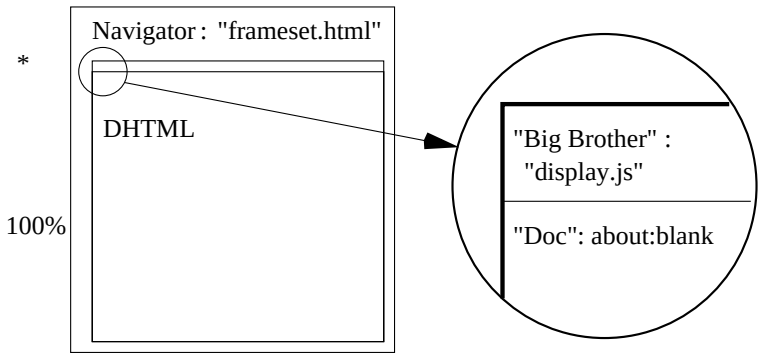


Figure 18: Using frames, first version.

that creates input arrays and variable initialisations for a JavaScript that does the rendering of the page. Only the popup menus showing up at the annotation points are dynamic HTML elements. The result is amazingly fast !

7 Conclusions

The implementation of interaction using a standard that was originally meant just to browse static information requires some tricks, some of which are rather awkward. As result of this project exists a basic implementation of an annotation system which allows inline editing and full HTML annotations. A remarkable and quite unique feature is the client side dynamic page recreation, which in combination with a decent locking mechanism can help cutting down network traffic.

References

[1] `http://www.parc.xerox.com/istl/projects/sparrow/`

[2] `http://www.thirdvoice.com`

[3] `http://www.nyx.net/ gthompso/quine.htm`

[4] Knuth, D. (1973). The Art of Computer Programming. Vol.3. Addison-Wesley.

[5] `http://dblab.kaist.ac.kr/ redstone/DL/html/brio_www95.html`

I am a 25 years old student of physics from Berlin, Germany. At the moment I am writing my diploma work in the Max-Born-Institut about the computation of the ionization of simple atoms in strong laser pulses. The band “Hennecke Beat” that I sing in exploits the idea of rock for peace but the lyrics express a strong social disorientation.

I want to thank my supervisor Dr. Mario Antonioletti for his support, also what my english maid had been if I haven’t had him.

A tree.pl

```
package Tree;

### GROW THE WISDOM TREE

$total=0;                              # total number of children with content (no
@level=();
@offspring_end=();

sub get_tree{
  my ($text,$begin,$end,$path,$author,$time)=@_;
  my $children_ref;
  my $child_ref;
  my $child_path_ref;
  my $ann;
  my $first;
  my $loc_author;
  my $loc_time;
  my @result=();
  $first=$total;
  while ($text =~ m/$begin/s){
    $loc_author=$1;
    $loc_time=$2;
    if ($‘){
      $child_path_ref=new_ref(@$path,scalar(@result));
      $child_ref = new_ref($‘,0,$child_path_ref,$autor,$time,$total);
      push(@result, $child_ref);
      push(@level,scalar(@$path));
      $total++;
    }
    $text=$’;
    $child_path_ref=new_ref(@$path,scalar(@result));
    $ann=get_annotation($text,$begin,$end);
    $children_ref=new_ref(get_tree($ann->[0],$begin,$end,$child_path_ref,$loc_au
    $child_ref = new_ref(’’,$children_ref,$child_path_ref,$loc_author,$loc_time,
    push(@result, $child_ref);
    $text=$ann->[1];
  }
  if ($text){                                                  # part behind the las
    $child_path_ref=new_ref(@$path,scalar(@result));
    $child_ref = new_ref($text,0,$child_path_ref,$author,$time,$total);
    push(@result, $child_ref);
    push(@level,scalar(@$path));
    $total++;
  }
  @offspring_end[$first]=$total-1;
  return @result;
```

```
}

sub get_annotation{                                    # gives you the entire r
  my ($text,$begin,$end)=@_;                          # matching <!-- ANN --> and
  my $depth=1;                                         # further parsing
  my $storage;
  my $result;

  while ($text =~ m/$end/s){
    if ($text =~ m/($begin)|($end)/s){
      if ($1){ # beginning of an annotation
$depth++;
$storage.=$‘.$&;
$text=$’;
      }
      else { # end of an annotation
$depth--;
$storage.=$‘;
$text=$’;
if ($depth==0){
  $result = new_ref($storage,$text);
  return $result;
}
$storage.=$&;
      }
    }
  }
  die ("file corrupt !");
}

sub path_string{               # concats the path with commas (for the GET strin
  $string=shift(@_);
  foreach $item (@_){
    $string.=",$item";
  }
  return $string;
}

sub parent_path_string{        # concats the parents path with commas (for the G
  pop(@_);
  $string=shift(@_);
  foreach $item (@_){
    $string.=",$item";
  }
  return $string;
}

sub get_text{                  # extracts all text in an annotation, (content of
```

```
  my $my_text='';
  foreach $item (@_){
    my ($text,$children_ref,$path_ref)=@$item;
    if (!$children_ref){
      $my_text .= $text;
    }
  }
  return $my_text;
}

sub new_ref{                          # creates a reference to a new array
my @array=@_;
return \@array;
}

sub get_parts{                        # gives you the contents of the HEAD, the B
  if( $_[0] =~ m:<HEAD>(.*)</HEAD>.*<BODY(.*?)>(.*)</BODY>:sgi){
    return ($1,$2,$3);
  }
  else {
    die ("No Body found !");
  }
}

1;
```

B proc_add.pl

```
#!/usr/local/bin/perl -w
require ’./tree.pl’;
require ’./globals.ph’;

use CGI;

$query      = new CGI;
$infile     = $query->param(’file’);
$path      = $query->param(’path’);
$par        = $query->param(’par’);
$state      = $query->param(’state’);
$new_text   = $query->param(’text’);

$new_time   = localtime;
$new_author = $ENV{’REMOTE_USER’};

$outfile    = $infile;
$|          = 1;        # flush !!!!

open(INFILE, $infile);
undef $/;
$all=<INFILE>;  # slurp !!!!
close(INFILE);

open(OUTFILE,">$outfile");
@tree=Tree::get_tree($all,$ANN,$ENDANN,[],’’,’’);
change_tree(@tree);
close(OUTFILE);

print "Location: $Cgi_bin/show.pl?file=$outfile&state=$state\n\n";

sub change_tree{
  my $path_string;
  foreach $item (@_) {
    ($text,$children_ref,$path_ref,$loc_author,$loc_time)=@$item;
    $path_string=Tree::path_string(@$path_ref);

    $author=$loc_author;
    $time=$loc_time;

    if (!$children_ref){ # content
      if ($path_string eq $path){
$text = insert_text($text,$par,$new_text,$new_author,$new_time);
      }
      print OUTFILE $text;
    }
```

```
    else {               # annotation
      print OUTFILE eval "qq^$ANNOUT^";
      change_tree(@$children_ref);
      print OUTFILE eval "qq^$ENDANNOUT^";
    }
  }
}

sub insert_text{         # this sub contains all the pattern matching for findin
  my ($text,$ins_par,$new_text,$author,$time) = @_;
  my $helper='';
  my $ready=0;
  my $par=0;

  while($text =~ m:$Anns:){
    print CHECK $&.$nl;
    $helper .= $`;
    $text = $';

    if ($par == $ins_par){
      $helper .= eval("qq^$ANNOUT^").$new_text.eval("qq^$ENDANNOUT^");
      $ready=1;
    }
    $helper .= $&;
    $par++;
  }
  if   (!$ready){
    return $helper.$text.eval("qq^$ANNOUT^").$new_text.eval("qq^$ENDANNOUT^"); }
  else          {
    return $helper.$text; }
}
```

C proc_edit.pl

```
#!/usr/local/bin/perl -w
require ’./tree.pl’;
require ’./globals.ph’;

use CGI;
$query    = new CGI;
$infile   = $query->param(’file’);
$path    = $query->param(’path’);
$action   = $query->param(’todo’);
$state    = $query->param(’state’);
$new_text = $query->param(’text’);

$path =~ /(.*),0/;
$parent_path = $1;
if (!$state) { $state=’’; }

$new_time = localtime;
$new_author = $ENV{’REMOTE_USER’};

$outfile= $infile;
$|      = 1;  # flush.

open(INFILE, $infile);
undef $/;
$all=<INFILE>;  # slurp !!!!
close(INFILE);

open(OUTFILE,">$outfile");
@tree=Tree::get_tree($all,$ANN,$ENDANN,[]);
change_tree(@tree);
close(OUTFILE);

print "Location: $Cgi_bin/show.pl?file=$outfile&state=$state\n\n";

sub change_tree{
  my $path_string;
  foreach $item (@_) {
    ($text,$children_ref,$path_ref,$author,$time)=@$item;
    $path_string=Tree::path_string(@$path_ref);

    if (!$children_ref){  # content
      if ($path_string eq $path){
if ($new_text){
  print OUTFILE $new_text; }}
      else {
if (Tree::parent_path_string(@$path_ref) ne $path) { # "collected" text does not
```

```
  print OUTFILE $text; }}}
    else {                          # annotation
      if ($path_string eq $parent_path){
$author = $new_author;
$time   = $new_time; }
      if ($path_string ne $parent_path || $action eq "edit"){
print OUTFILE eval "qq^$ANNOUT^";
change_tree(@$children_ref);
print OUTFILE eval "qq^$ENDANNOUT^";
      }
    }
  }
}
```

D show.pl

```
#!/usr/local/bin/perl -w
require ’./tree.pl’;
require ’./globals.ph’;

use CGI;
$query = new CGI;
$infile = $query->param(’file’);
$state  = $query->param(’state’);
$userid = $ENV{’REMOTE_USER’};
$new_time = localtime;

open(CHECK,">>logfile");
  print CHECK "show : $userid $new_time\n";
close(CHECK);

$alert = $query->param(’alert’);
if (!$count) {$count = 0;}
if (!$state) {$state = ’’;}

$|      = 1;  # flush.
print "Content-type: text/html\n\n";

if (!$infile) { $infile = $Defaultdoc;}
$infile = cripple($infile);

open(INFO, $infile);
undef $/;
$all=<INFO>;  # slurp !!!!
close(INFO);

($head,$bodytag,$body)=Tree::get_parts($all);           # splits up the document
$body = insert_scriptcalls($body);                      # makes shure, links are
@tree=Tree::get_tree($body,$ANN,$ENDANN,[],’’,’’);      # builds the tree
print ’<HTML><HEAD>
<META HTTP-EQUIV="expires" CONTENT="0">
<SCRIPT><!--
var was_me=1;
function framedoctor(){ if (was_me) was_me=0; else top.history.go(-1);}
//-->
</SCRIPT>’;
$head =~ s/\s+/ /g;
print $head;
print ’<SCRIPT>’;
print qq^var Head=’$head’;^.$nl;
print qq^var BodyTag=’$bodytag’;^.$nl;
print "
```

```
function out(doc,text){ doc.writeln(text); }
function ou(doc,text){ doc.write(text); }

var nrPatches      = $Tree::total;
var File           = '$infile';
var procEditScript = '$Cgi_bin/proc_edit.pl';
var procAddScript  = '$Cgi_bin/proc_add.pl';
var killScript     = '$Cgi_bin/proc_edit.pl';
var Anns           = new Array('$JSAnns');

var state          = '$state';
if (state.length > 0)
  var param        = state.split(',');
else
  var param        = new Array;

var c              = new Array(nrPatches);".$nl;
print '//  text , author , time , path , permission , level , offend , patches '
print_tree(\@tree,'');
print '
window.captureEvents(Event.RESIZE);
</SCRIPT>
</HEAD>
<FRAMESET ROWS="*,26" BORDER="0" onLoad="javascript:top.BB.main();">
    <FRAME NAME="DOC" SRC="http://www.epcc.ed.ac.uk/~presber/loading.html" SCROL
    <FRAME NAME="BB" SRC="http://www.epcc.ed.ac.uk/~presber/bb.html" SCROLLING="
</FRAMESET>
</HTML>';

#=============================================================================

sub print_tree{
  my ($mytree, $auth, $editable)=@_;
  my $path_string;
  my $owner;
  my $offend;
  my $first      = 1;
  my $permission = 0;
  my $patch_list = get_patch_list($mytree);
  foreach $item (@$mytree) {
    ($text,$children_ref,$path_ref,$author,$time,$number)=@$item;
    $path_string=Tree::path_string(@$path_ref);
    if (!$children_ref){                  # content
      # $outtext  = insert_marks($text,$number);
      if (!$first){ $patch_list='';}
      if ($editable && $first){ $permission=1; } else { $permission=0; }
      $text =~ s/\n/<<N>>/g;
      $text =~ s/\s+/ /g;
```

```
      $text =~ s/<<N>>/\\n/g;
      $text =~ s/"/\\"/g;
      $text =~ s/'/\\'/g;
      print 'c['.$number,']=new Array("'.$text.'","'.$auth.'","'.$time.'","';
      print Tree::path_string(@$path_ref).'"';
      if ($Tree::offspring_end[$number]) {$offend=$Tree::offspring_end[$number];
      print ','.$permission.','.$Tree::level[$number].','.$offend.',"'.$patch_li
      if ($first){ $first=0;}
    }
    else { # annotation
      $owner = ($author eq $userid || !$author);
      print_tree($children_ref,$author,$owner);
    }
  }
}

sub get_patch_list{
  my ($tree) = @_;
  my $result='';
  my $first=1;
  my ($text,$children_ref,$path_ref,$author,$time,$number);
  foreach $item (@$tree) {
    ($text,$children_ref,$path_ref,$author,$time,$number)=@$item;
    if (!$children_ref){                  # content
      if ($first){
$first=0;
$result=$number; }
      else { $result.=",$number";}
    }
  }
  return $result;
}


sub insert_marks{                      # inserts the annotation points at lines
  my ($text,$nr)=@_;
  my $helper='';
  my $par=0;

  while($text =~ m:$Line_anns|$Par_anns:si){
    if ($') {
      $par++;
      $helper .= $`.add_ref($nr,$par).$&;
    }
    $text = $';
  }
  $helper .= $text;
  return $helper;
```

```
}

sub insert_scriptcalls{
  my($text)  = @_;
  my $helper = '';
  my $one    = '';
  my $rest   = '';
  while ($text =~ m^<A.*?HREF="(.*?)">^i){
    $helper.= $`'<A HREF="';
    $one=$1;
    $rest=$';
    if ($one =~ m^http://|www\.|mailto:^){
      $helper .= $one.'" TARGET="_top" >';
    } else {
      $helper .= $Cgi_bin.'/show.pl?file='.cripple($one).'" TARGET="_top"> ';
    }
    $text = $rest;
  }
  $helper .= $text;
  return $helper;
}

sub insert_script{
  my ($text,$nr_children)=@_;
  my $i=0;
  my $levels;

  if (scalar(@Tree::level)==1){$levels="0,0"}
  else{
    $levels=shift(@Tree::level);
    foreach $item(@Tree::level){
      $levels .= ','.$item;
    }
  }
  if (scalar(@Tree::offspring)==1){$offspring="0,0"}
  else{
    $offspring = shift(@Tree::offspring_end);
    foreach $item(@Tree::offspring_end){
      if (!$item){ $item=0 }
      $offspring .=  ','.$item;
    }
  }

  return $text;
}

### HTML-SUBS
```

```
sub add_ref{
  my ($nr,$par,$p)=@_;
  my $reply;
  if (!$par){
    return '<A CLASS="ann" HREF="javascript:add('.$nr.',0)">'.$Add_pic.'</A>';
  } else {
    if ($p){
      $addP_nr[$nr]++;
      $reply  = '<SPAN ID="addP'.$nr.'_'.$addP_nr[$nr].'"><A CLASS="ann" HREF="j
      $reply .= $Add_pic.'</A></SPAN>'.$nl;
    } else {
      $addL_nr[$nr]++;
      $reply  = '<SPAN ID="addP'.$nr.'_'.$addL_nr[$nr].'"><A CLASS="ann" HREF="j
      $reply .= $Add_pic.'</A></SPAN>'.$nl;
    }
    return $reply;
  }
}

sub cripple{                    # cripples the string after "?file=" so the docu
  my ($file)=@_;
  if ($file =~ m^/^){
    $file =~ m^.*/(.*.html)^;
    $file = "$Docroot/$1";
  } else {
    $file = "$Docroot/$file";
  }
  return $file;
}
```

E bb.html

```
<HTML>
  <HEAD>
    <STYLE>
      #OUTPUT {position:absolute; top:4; left:300; visibility: visible; font-fam
</STYLE>
    <SCRIPT><!--
reloaded=document.search;
var PicWidth    = 20;
var PicHeight   = 20;
var nrMenus     = 2;
var nrMenuItems = 5;
var Wwwroot     = "http://www.epcc.ed.ac.uk/~presber/";
var Imageroot   = Wwwroot+"images/";
show1   = new Image(); show1.src=Imageroot+"show_ann.gif";
show2   = new Image(); show2.src=Imageroot+"hide_ann.gif";
ann1 = new Image(); ann1.src=Imageroot+"annotate.gif";
ann2 = new Image(); ann2.src=Imageroot+"noannotate.gif";

// DECLARATIONS

if (top.param[1]) var scrollTo  = top.param[1]; else var scrollTo  = 0;
if (top.param[0]) var showLevel = top.param[0]; else var showLevel = 0;
var lastScroll=0;
var lastLevel;
var lastState;
var allState="";
var d;
var MenuItem = new Array(nrMenuItems);
MenuItem[0]  = new menuItem(Imageroot+'_left.gif','hideMenu',0);
MenuItem[1]  = new menuItem(Imageroot+'_up.gif','hide',0);
MenuItem[2]  = new menuItem(Imageroot+'_down.gif','edit',1);
MenuItem[3]  = new menuItem(Imageroot+'_kill.gif','kill',1);
MenuItem[4]  = new menuItem(Imageroot+'_a.gif','showAnnPoints',0);
var Trigger  = new Array(nrMenus);
Trigger[0]   = Imageroot+"info.gif";
Trigger[1]   = Imageroot+"right.gif";
var Opener   = new Array(nrMenus);
Opener[0]    = Imageroot+"greydot.gif";
Opener[1]    = Imageroot+"dot.gif";

for (i=0; i<nrMenus; i++)
menu[i] = new menu("menu"+i,i,0);

var patch = new Array(top.nrPatches);
var c;
for (i=0; i<top.nrPatches; i++){
```

```
    c=top.c[i];
    patch[i] = new child(i,c[0],c[1],c[2],c[3],c[4],c[5],c[6],c[7]);
}

function main(){
    d=top.DOC.document;
    d.open("text/html");
    if (top.param[0])
document.images["showpic"].src=show2.src;
    redraw();
}

// function definitions

function check_visibilities(){
    for (var i=0; i<patch.length; i++)
if (patch[i].outVisible && patch[i].ownVisible && patch[i].level>0)
    for (var j=i+1; j<=patch[i].offend; j++){
if (patch[j].level==patch[i].level+1)
    patch[j].outVisible=1;
if (showLevel<patch[j].level)
    showlevel=patch[j].level;
    }
    return 1;
}

function scrollmeTo(i){
    if (i==-1 && top.DOC.document.images['annProcPic'])
lastScroll = top.DOC.document.images['annProcPic'].y-50;
    if (i>0 && top.DOC.document.images['annPic' + i ])
        lastScroll = top.DOC.document.images['annPic' + i ].y-50;
    top.DOC.scroll(0,lastScroll);
}


function redraw(){
    check_visibilities();
    out('<HTML><HEAD>');
    out('<META HTTP-EQUIV="expires" CONTENT="0">');
    out('<STYLE TYPE="text/css">');
    out('#menu0 {position:absolute; visibility: hidden;}');
    out('#menu1 {position:absolute; visibility: hidden;}');
    out('.ann   {font-family:sans-serif; font-size:medium;}');
    out('.name   {font-family:sans-serif; font-style:italic; font-size:x-small;}
    out('</STYLE>');
    out(top.Head);
    out('</HEAD>');
    out('<BODY '+ top.BodyTag + ' onLoad="top.BB.scrollmeTo('+ top.BB.scrollTo +
```

```
    out('<A NAME="ANN"></A>');
    var lastShown = 0;
    for (var i=0; i<patch.length; i++){
c=patch[i];
level      = c.level;
outVisible = c.outVisible;
ownVisible = c.ownVisible;

if (outVisible && level<=showLevel && (allState=="" || i==scrollTo) ) {
    if (level>0 && (ownVisible || ( lastLevel !=0 && ((ownVisible != lastOwnVisi
out('<BR>');
for (j=1; j < level; j++)
    out('   ');
    }
    if (i>0 && top.BB.scrollTo==i)
ou('<IMG SRC="'+Imageroot+'pointer.gif" HEIGHT="'+PicHeight+'">');
    c.drawBt();
    lastLevel        = level;
    lastOwnVisible   = ownVisible;
}
if (outVisible && ownVisible && level<=showLevel){
    if (level>0)
out('<SPAN CLASS="ann">');
    switch (c.state){
    case "edit"      : out(c.editHTML())      ; break;
    case "select"    : out(c.annHTML(-1))     ; break;
    case "annotate"  : out(c.annHTML(c.par))  ; break;
    default       : out(c.text);
    }
    if (level>0)
out('</SPAN>');
}
    }
    out(menuText(0));
    out(menuText(1));
    out('</BODY></HTML>');
    d.close();
}


function out(text){
    top.out(d,text);
}

function ou(text){
    top.ou(d,text);
}
```

```
function clearInfo(){
    with (document.OUTPUT.document){
open();
close();
    }
}

function toggleShow(){
    if (showLevel){
showLevel=0;
document.images["showpic"].src=show1.src;
scrollTo=0;
redraw();
    }
    else{
showLevel=1000;
document.images["showpic"].src=show2.src;
redraw();
    }
}

function toggleAnnotate(){
    if (allState == "select"){
document.images["annpic"].src=ann1.src;
allState="";
patch[0].sapCancel();
    }
    else{
document.images["annpic"].src=ann2.src;
allState="select";
patch[0].showAnnPoints();
    }
}

function hideMenu(level){
    if (!level)                // hide ANY open menu
for(i=0; i<nrMenus; i++){
    top.DOC.document.layers[menu[i].name].visibility = 'hidden';
    menu[i].visible=0;
}
    else {                     // hide menu level
var mn=menu[level];
top.DOC.document.layers[mn.name].visibility = 'hidden';
mn.visible=0;
    }
}

function showMenu(level,nr){
```

```
    var mn = menu[level];
    var counter=1;
    hideMenu();

    myImg = top.DOC.document.images["annPic"+nr];
    myLayer = top.DOC.document.layers[mn.name];

    for (i=1; i<nrMenuItems ; i++)
if (mn.level >= MenuItem[i].level){
    myLayer.document.links[counter].href='javascript:top.BB.patch['+nr+'].' +  M
    counter++;
}
    with (myLayer){
left       = myImg.x;
top        = myImg.y;
visibility = "visible";
    }
    mn.visible=1;
}

function menuText(level){
    var result='<DIV ID="menu' + level + '">';
    var menuStr='';
    for (i=0; i<nrMenuItems ; i++)
if (level>=MenuItem[i].level){
    menuStr += '<A HREF="';
    if (i==0)
menuStr += 'javascript:top.BB.hideMenu('+ level +');';
    else
menuStr+='#';
    menuStr+='">';
    menuStr += '<IMG NAME="menuPic' + i + '" SRC="' + MenuItem[i].pic + '" ';
         menuStr += 'WIDTH="' + PicWidth + '" HEIGHT="' + PicHeight + '" BORDER=
    menuStr += '</A>';
}
    result += menuStr;
    result +='</DIV>';
    return result;
}

function buttonText(nr,perm,shown){
    if (!shown){
var result  ='<A HREF="javascript:top.BB.patch['+ nr +'].show();" ';
result += 'onMouseover="javascript:top.BB.patch['+nr+'].info()" onMouseout="java
        result += '<IMG NAME="annPic' + nr + '" SRC="' + Opener[perm] + '" BORDE
result += '<SPAN CLASS="name">'+patch[nr].author+'</SPAN>';
    }
    else {
```

```
var result  = '<A HREF="javascript:top.BB.showMenu(' + perm +','+ nr + ');" ';
result += 'onMouseover="javascript:top.BB.patch['+nr+'].info()" onMouseout="java
result += '<IMG NAME="annPic' + nr + '" SRC="'+Trigger[perm]+'" BORDER="0" HEIGH
    }
    return result;
}

// "CONSTRUCTORS" AND "MEMBERFUNCTIONS"
//OBJECT CHILD

function child(i,text,author,time,path,permission, level, offend, patches){
    this.id         = i;          // same as field index
    this.author     = author;
    this.time       = time;
    this.text       = text;       // actual content
    this.path       = path;       // path for calling the scripts
    this.level      = level;      // level for indent
    this.permission = permission; // 0 r, 1 rw
    this.offend     = offend;     //"offspring end" last patch that is part of t
    if (patches.length>0)
this.patches = patches.split(',');            // annotated text consists of >0
    else
this.patches = new Array ();                  // empty, only for reference
    this.button     = this.patches.length*this.level;
    switch(level){
    case 0:{
this.outVisible = 1;                      // visibility of parent
this.ownVisible = 1;
break;}
    case 1:{
this.outVisible = 1;
this.ownVisible = 0;
break;}
    default:{
this.outVisible = 0;
this.ownVisible = 0;
break;}
    }
    if (scrollTo==i){
this.outVisible = 1;
this.ownVisible = 1;
    }
    this.state        = "";
    this.last         = 0;
    this.par          = 0;
    // member functions
    this.edit         = edit;
    this.eCancel      = eCancel;
```

```
    this.annotate      = annotate;
    this.aCancel       = aCancel;
    this.kill          = kill;
    this.hide          = hide;
    this.show          = show;
    this.drawBt        = drawBt;            // draws the menu button
    this.info          = info;
    this.editHTML      = editHTML;
    this.showAnnPoints = showAnnPoints;
    this.sapCancel     = sapCancel;
    this.annHTML       = annHTML;
    this.textArea      = textArea;
}

function textArea(action,state,todo,text,cancelFunc){
    var result='';
    result+='<SPAN CLASS="ann"><FORM ACTION="'+action+'" TARGET="_top" METHOD="G
    result+='<INPUT TYPE="HIDDEN" NAME="todo"  VALUE="'+todo+'">';
    result+='<INPUT TYPE="HIDDEN" NAME="state" VALUE="'+state+'">';
    result+='<INPUT TYPE="HIDDEN" NAME="file"  VALUE="'+top.File+'">';
    result+='<INPUT TYPE="HIDDEN" NAME="path"  VALUE="'+this.path+'">';
    result+='<INPUT TYPE="HIDDEN" NAME="par"   VALUE="'+this.par+'">';
    result+='<TEXTAREA NAME="text" COLS="50" ROWS="5" WRAP="VIRTUAL">';
    result+= text;
    result+='</TEXTAREA><BR>';
    result+='<INPUT TYPE="SUBMIT" VALUE="'+todo+'"> ';
    result+='<INPUT TYPE="BUTTON" VALUE="cancel" onClick="top.BB.patch['+this.id
    result+='</FORM></SPAN>';
    return result;
}

function editHTML(){    // inserts an edit field
    var text='';
    for (j=0; j<this.patches.length; j++ )
text+=patch[this.patches[j]].text;
    return this.textArea(top.procEditScript,this.level+','+this.id,"edit",text,"
}

function annHTML(p){  // inserts an annotate button OR the textfield
    var text=this.text;
    var result='';
    var i;
    var pos;
    var tag='';
    var loc_pos;
    var count=0;
    if (this.button){
patch[this.patches[this.patches.length-1]].last=this.id;
```

```
    }
    while(text.length>0){
tag='';
pos = text.length;
for (i=0; i<top.Anns.length; i++){
    loc_pos = text.indexOf(top.Anns[i]);
    if ((loc_pos!=-1) && (loc_pos < pos)){
pos = loc_pos;
tag = top.Anns[i];
    }
}
result += text.substring(0,pos);
if (tag.length>0 || this.last )
    if (p==-1){
result += '<A HREF="javascript:top.BB.patch['+this.id+'].annotate('+count+');">'
result += '<IMG SRC="'+Imageroot+'_a.gif'+'" WIDTH="' + PicWidth + '" HEIGHT="'
result += '" BORDER="0"></A>'+tag;
    }
    else {
scrollTo=-1;
if (p==count){
    result+='<BR>';
    for (var j=1; j <= this.level; j++)
result+= '   ';
    result +='<IMG SRC="'+Trigger[1]+'" NAME="annProcPic" WIDTH="' + PicWidth +
    result += '" BORDER="0"></A>';
    result +=this.textArea(top.procAddScript,(this.level+1)+',+(this.id+1),"anr
}
else
    result +=tag;
    }
text = text.substring(pos+tag.length);
count++;
    }
    if (this.last){
result += '<BR><FORM><INPUT TYPE="button" VALUE="cancel" onClick="top.BB.patch['
this.last=0;
    }
    return result;
}


function info(){
    with (document.OUTPUT.document){
open();
write('<FONT SIZE="+1" COLOR="red" FACE="sans-serif">');
write(this.author+" "+this.time);
write('   : "'+(this.text.replace('<','&lt;')).substring(0,30)+'..."')
```

```
write('</FONT>');
close();
window.status=this.author+" "+this.time;
    }
}

function edit(){
    if (this.state == "editing") return;
    this.state="edit";
    allState="edit";
    scrollTo=this.id;
    redraw();
}

function eCancel(){
    this.state="";
    allState="";
    redraw();
}

function kill(){
    var count=top.count+1;
    if (confirm("Are you sure you want to delete the annotation\n\n\""+this.text
top.location.href=top.killScript+"?file="+top.File+"&path="+this.path+"&state=1,
    else
hideMenu();
}

function showAnnPoints(){
    for (count = 0; count< this.patches.length; count ++)
patch[this.patches[count]].state = "select";
    scrollTo=this.id;
    redraw();
}

function sapCancel(){
    for (count = 0; count< this.patches.length; count ++)
patch[this.patches[count]].state = "";
    allState="";
    scrollTo=0;
    if (this.level==0)
document.images["annpic"].src=ann1.src;
    redraw();
}

function annotate(par){
    for (count = 0; count< top.nrChildren; count ++)
patch[count].state="";
```

```
    this.state = "annotate";
    allState="annotate";
    this.par=par;
    scrollTo=-1;
    redraw();
}

function aCancel(){
    this.state="";
    allState="";
    if (this.level==0)
document.images["annpic"].src=ann1.src;
    redraw();
}

function hide(){
    this.ownVisible = 0;
    for (var count = this.id + 1; count<= this.offend; count ++)
patch[count].outVisible=0;
    for (count = 0; count< this.patches.length; count ++)
patch[this.patches[count]].ownVisible=0;
    scrollTo=this.id;
    redraw();
}

function show(){
    this.ownVisible = 1;
    for (count = this.id + 1; count<= this.offend; count ++)
if (patch[count].level==this.level+1)
    patch[count].outVisible = 1;
    for (count=0; count< this.patches.length; count++)
patch[this.patches[count]].ownVisible = 1;
    scrollTo=this.id;
    redraw();
}

function drawBt(){
    if (this.outVisible && this.button>0)
out(buttonText(this.id,this.permission,this.ownVisible));
}

// OBJECT : MENUITEM

function menuItem(pic,func,level){
    this.pic=pic;
    this.func=func;
    this.level=level;
}
```

```
// OBJECT : MENU

function menu(name,level,visible){
    this.level=level;
    this.name=name;
    this.visible=0;
}

//-->
    </SCRIPT>
  </HEAD>
  <BODY BGCOLOR="#FFFFFF"><A HREF="javascript:top.BB.toggleShow();"><IMG NAME="s
<A HREF="javascript:top.BB.toggleAnnotate();"><IMG NAME="annpic" SRC="http://www
    <SPAN ID="OUTPUT"></SPAN><NOSCRIPT>Sorry, you need JAVASCRIPT turned on to r
  </BODY>
</HTML>
```

F loading.html

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <STYLE>
      BODY,H1 {font-family:helvetica,sans-serif; background-color:white;}
    </STYLE>
    <title>loading</title>
    <SCRIPT> top.framedoctor(); </SCRIPT>
  </head>
  <body>
    <h1>loading...</h1>
    <hr>
  </body>
</html>
```

G globals.ph

```
### PATHS
$Docroot = '../html/mpi-course';
$Defaultdoc     = 'index.html';
$Cgi_bin = 'http://www.epcc.ed.ac.uk/cgi-bin/presber';
$Wwwroot = 'http://www.epcc.ed.ac.uk/~presber';

### SYNTAX

$ANN = '\n?<!-- ANN AUTH="(.*?)" TIME="(.*?)" -->\n?';
$ANNOUT = '\n<!-- ANN AUTH="$author" TIME="$time" -->';
$ENDANN = '\n?<!-- END ANN -->\n?';
$ENDANNOUT = '<!-- END ANN -->';

@Not_allowed    = ('<!--.*?-->');
@Must_be_closed = ('<H(.?)>','<B>','<P>');
$Anns    = '<BR>|<br>|</LI>|</li>|<P>|<p>|</P>|</p>|<UL>|</ul>|<TABLE>|<table>|<
$JSAnns = $Anns;
$JSAnns =~ s/\|/','/g;

### HTML-STYLES
$Symbolsize  = 16;
$Buttonsize  = $Symbolsize;
$Fontsize = $Symbolsize;

### GIMMICKS

$nl = "\n";
$tab = "\t";

sub nullofy{
  my ($number)=@_;
  if (!$number){$number=0};
  return $number;
}

sub alert{
  my ($message)=@_;
  print '<SCRIPT>alert("'.$message.'");</SCRIPT>';
}

sub script{
  my ($text)=@_;
  print '<SCRIPT LANGUAGE="javascript">'.$text.'</SCRIPT>';
}
```