Brussels Free University
Faculty of Sciences

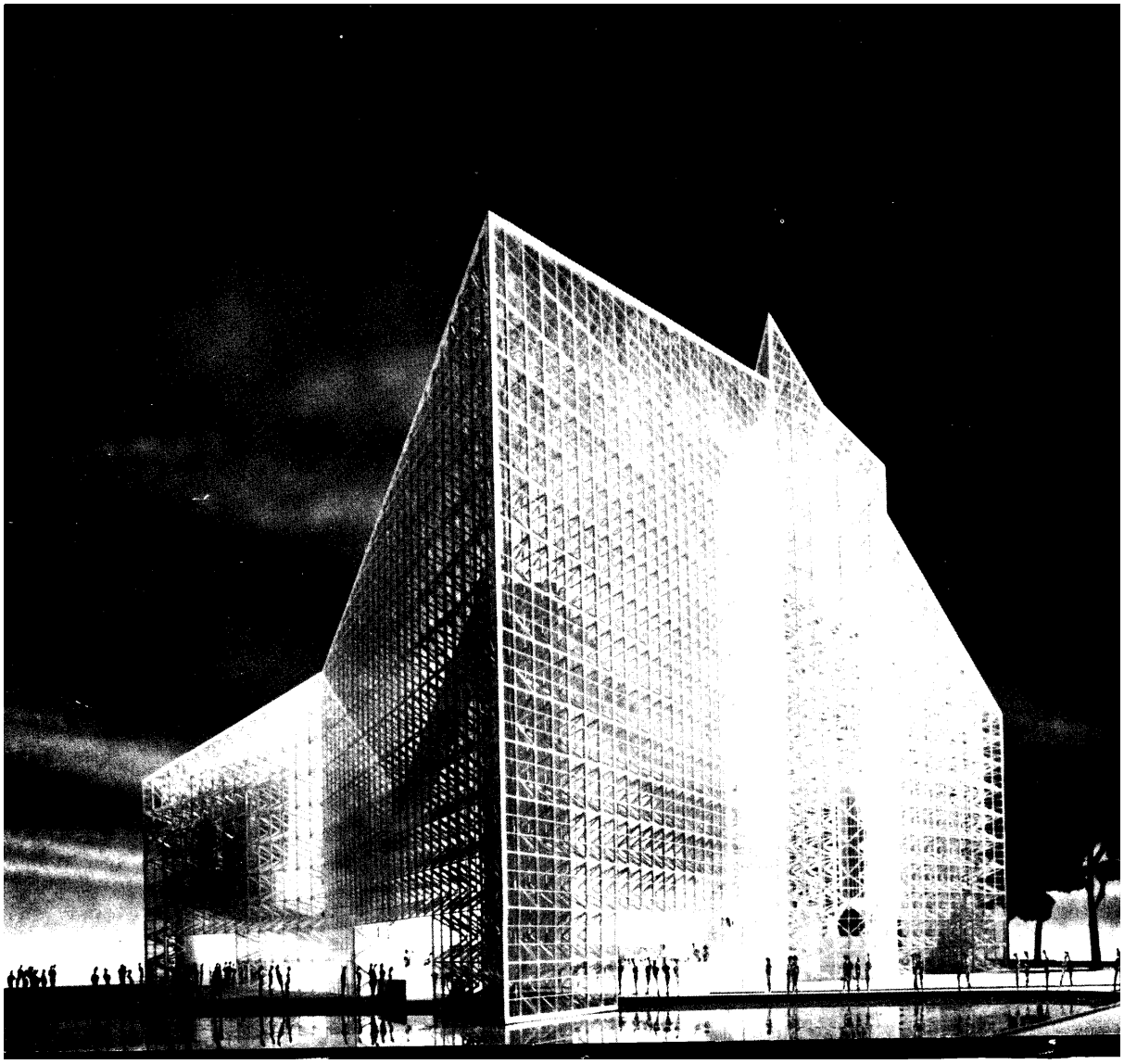
Academic Year 1994 - 1995

An Application Framework for HTML WEB Browsers

Thesis submitted in view of
acquiring the degree of
Master in Computer Science

By Luk Stoops

Promoter Prof. Theo D'Hondt



1. Contents

2. Acknowledgments

I like to thank:

- My **wife and children** for their patience and support.
- **Wim Lybaert, Mark Plas, Hans-Martin Mosner** for their help.
- **Theo D'Hondt** for his trust.

Luk Stoops
August 1995

I think that it's extraordinarilay important that we in computer science keep fun in computing. When it started out, it was an awful lot of fun. Of course, the paying customers got shafted every now and then, and after a while we began to take their complaints seriously. We began to feel as if we really where responsible for the successful, error-free perfect use of these machines. I don't think we are. I think we're responsible for stretching them, setting them off in new directions, and keeping fun in the house.

Alan J. Perlis

3. Introduction

In the world of the WEB, where Darwin-like selection mechanisms are active, WWW browsers need to change continuously to keep up with the growing demands of web users.

Not only for the new media standard's (GIF, JPG, AU, WAV, JPEG ,VRML ...) where classic browsers needs their own external helper programs. One for each media and one for each hardware platform.
The explosion of the web facilities provokes an explosion of helper programs.
Today it is impossible to install on your system all possible helper applications. Finding the latest versions of all helper programs for all platforms is hopeless.
On top of that, in the advent of secure http protocols, it is expected that more and more websites will provide services for a limited group of customers.
These customers will need a customized browser to get access to the services offered.

Their is also still a lot to be done to help novice users to form their mental models of the WWW and to offer them navigating tools in the information spaces.

In this document we explore a web browser framework to build all kind of different browsers. The framework itself is build in the VisualWorks 2.0 smalltalk environment so that the resulting browsers are easily integrated, as a whole or partially, in existing applications and that existing applications are easily integrated in the browsers.

I discovered and learned a lot the last years, not least the existence of internet and the web. It was fun.

Software reuse does not happen by accident, even with object-oriented programming languages. System designers must plan to reuse old components and must look for new reusable components. The Smalltalk community practices reuse very successfully: The keys to successful software reuse are attitude, tools and techniques.
Ralph E. Johnson [i]

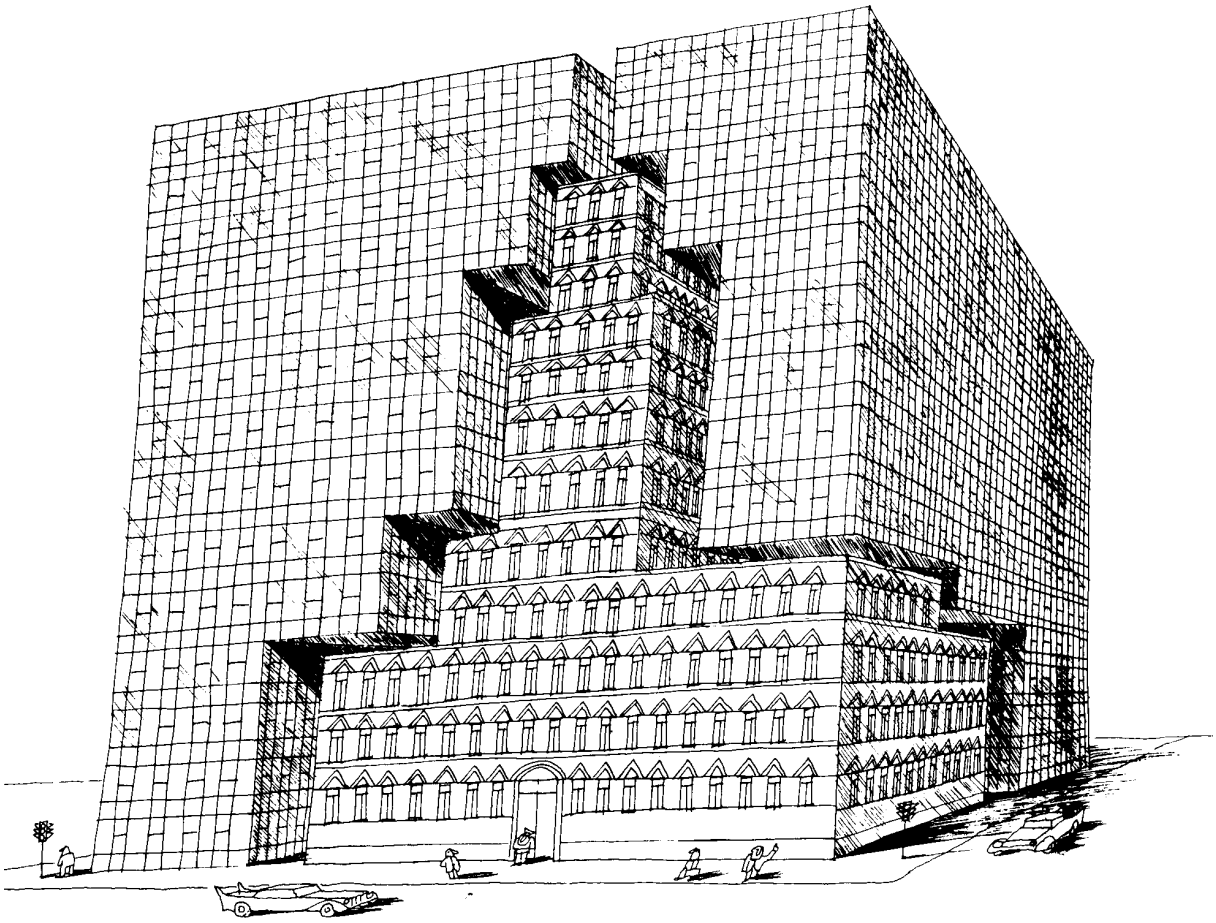


Figure 1 (from Bauherrlichkeit - München : Gabor Benedek)

4. Frameworks

A framework is a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuses at a larger granularity than classes.

The big building symbolizes the framework while the embedded building stands for the supplied user code. (figure 1)

The described framework evolved from the "one shot" web browser that was implemented during my specialization training this year.
Useful abstractions are usually designed from the bottom up, i.e. they are discovered, not invented. We create new general components by solving specific problems, and then recognizing that our solutions have potentially broader applicability.

Ralph E. Johnson suggested some rules to enhance the reusability of classes.
Much of them are applied here.

- Most of the case analysis was eliminated by sending the messages directly to the object who's class was checked.
- The number of arguments was reduced.
- The size of the methods was reduced.
- Abstract classes where introduced where appropriate.
- Most of the instance variables are now accessed via messages instead of accessing them directly.
- Subclasses are specializations.
- Large classes are split.

4.1 White-box Frameworks

An important characteristic of a framework is that the methods defined in the user code will often be called from within the framework itself, rather than from the user's application code. This characteristic is known as the Hollywood principle (don't call us we call you). (figure 2)

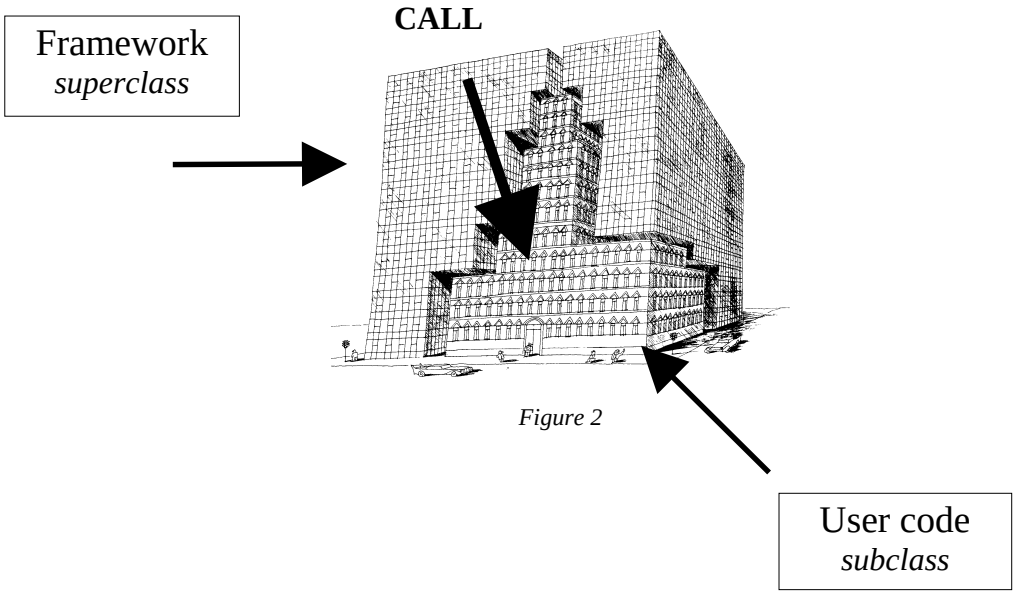


Figure 2

We call these white-box frameworks because their implementation must be understood to use them.
The Model View Controller controller class is an example of a white-box framework.
The major problem with such a framework is that every application requires the creation of many new subclasses.

4.2 Black-box Frameworks

Pluggable views are components of the Model View Controller framework that are called black-box frameworks because the pluggable views let controllers take the menus as parameters, thus greatly reducing the need to create new controller classes.

Black-box frameworks like the pluggable views are easier to learn to use than white-box frameworks, but are less flexible. Pluggable views are usually sufficient to describe user interfaces that display only text, but a web user who wants a more graphical user interface will have to use the original MVC framework.

Ideally, each framework will evolve into a black-box framework but many frameworks will not complete the journey from skeleton to black-box frameworks during their lifetimes.

White-box inheritance frameworks should be seen as a natural stage in the evolution of a system. Because they are a middle ground between a one-shot application and an abstract design.

4.3 The Smalltalk MVC Framework

The HTML WEB Browser framework is build upon the Model-View-Controller framework which is a white-box framework for constructing Smalltalk-80 user interfaces. A separate model class HtmlModel was created encapsulating al instance variables and methods that were independent of the user interface class.

5. Using a HTML-browser

Before diving in the reuse of the browser concept let's have a look at the different ways of using the example HTML browser.

- Start the browser via the **Resource Finder.** (figure 3)

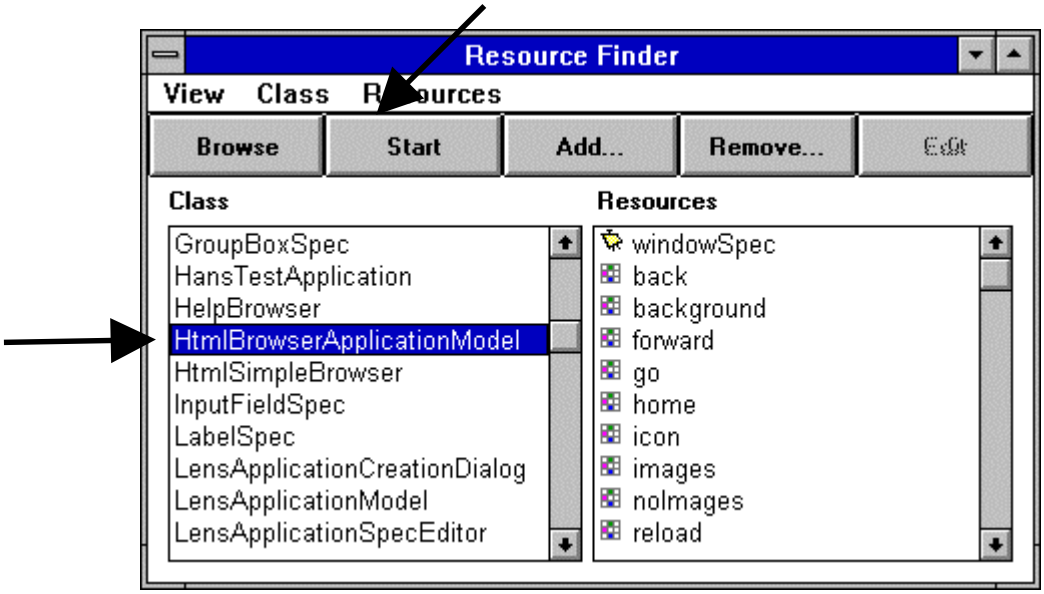


Figure 3

- Send the message **open**
  or **open:** 'url' to **HtmlBrowserApplicationModel**

  - **HtmlBrowserApplicationModel open**
  - **HtmlBrowserApplicationModel open:** 'file:///c|/luk/vub/html/luk/test3.htm'

All the methods referenced in this document are included in the Appendix.

The complete code is available via http://progwww.vub.ac.be/persons/lstoops/luk_home.htm

*Class - and method names are in **bold***

- Use the new added **WEB icon** in the Visual Launcher (figure 4)

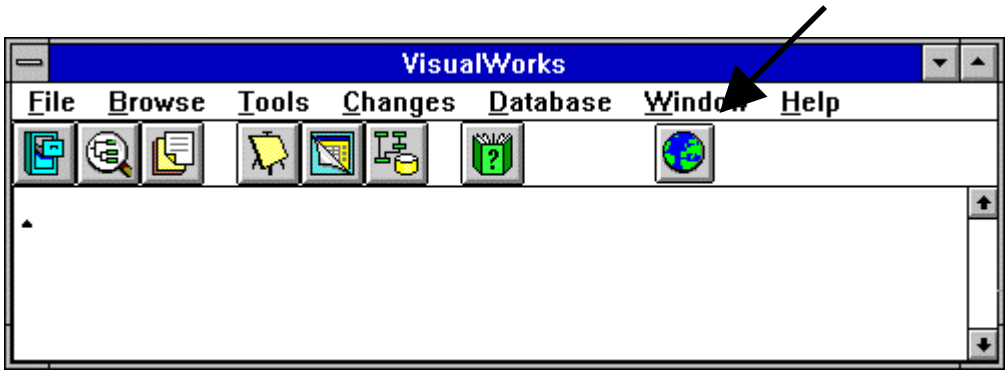


Figure 4

- Select in a **Workspace** a text in a valid url format and choose the new added yellow button menu-item '**as url**'. (figure 5)

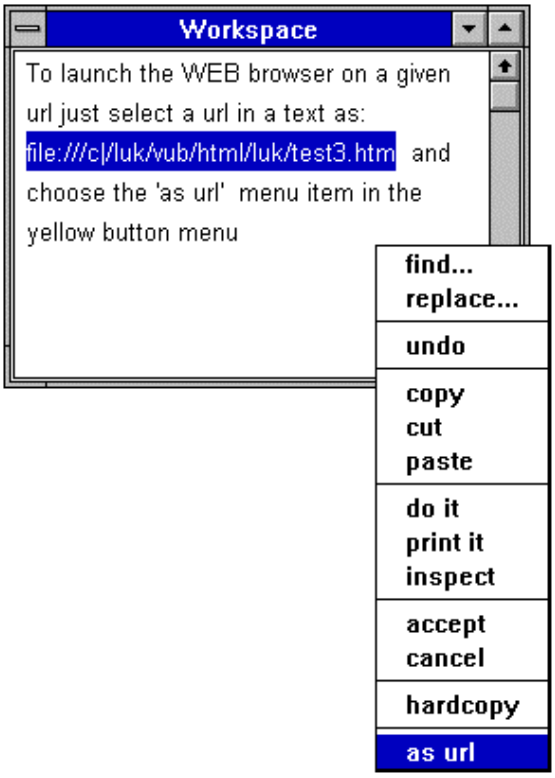


Figure 5

After sending the message **open** to **HtmlBrowserApplicationModel** a Browser window will open depending on the user settings for Window Placement (User placement or Automatic placement). (figure 6)

Figure 6

The design of the user interface is based on the results of a study, conducted at the Georgia Institute of Technology, about actual user behaviour, as determined from client-side log file analysis[ii] as presented at the third international www-congress in Darmstadt.

The most used actions are:

1. change url by clicking on a hyperlink. (16175)
2. change url using the Back button. (12632)
3. change url via hotlist. (2336)
4. open url. (1753)
5. reload current. (1507)

This is why the Back button becomes the biggest button of the interface and the yellow-mouse-button gives a menu limited by the most asked actions:

- Back
- Bookmarks
- open Location
- reload

The slider can be used to navigate in the history and making it extra easy to go back one or more url's.
The range of the slider is dynamically adapted to the number of url's visited.

The *Halt* button send a halt message to HtmlBrowserApplicationModel.

The *Test* button is used to activate test messages.

The *Closure* button can be used to load related url's in cache memory.

Hotlist, help, questions etc. are implemented as simple url calls.

The collapsed window is visualized by a WWW icon, (figure 7) instead of the classic Smalltalk icon. (figure 8)

Figure 7

Figure 8

The link to this icon can not be done in the **initialize** method since the builder is not yet available at that time.

A method **postBuildWith: aBuilder** is used to link the icon to the window.

6. Reusing a html-browser

The Visual Works Smalltalk environment makes it particularly easy to integrate the browser complete or partially in other new or existing applications.

It can be done in a few simple steps.

- **Edit the windowSpec of HtmlBrowserApplicationModel. (figure 9)**

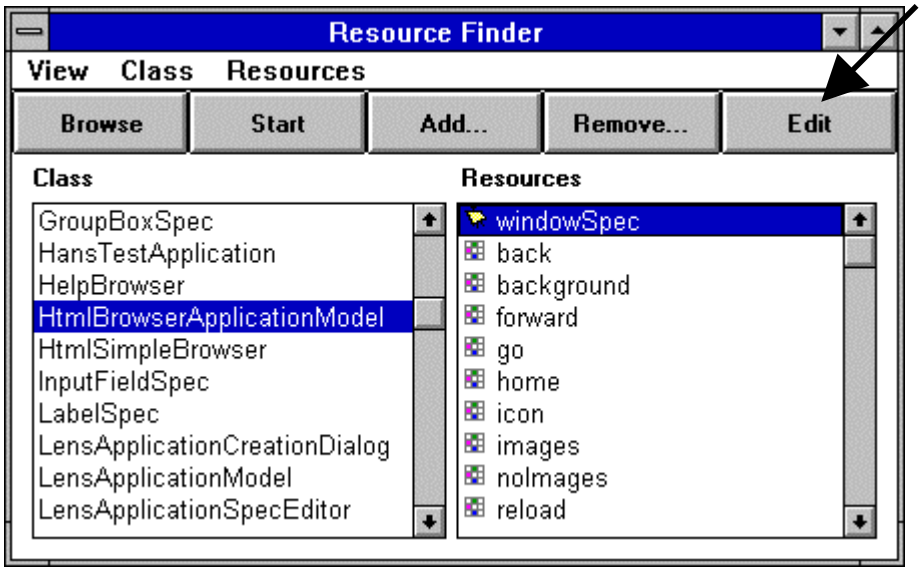


Figure 9

- **Select the views, buttons, texteditors etc. that needs to be reused in the existing application and copy them. For instance the home button. (figure 10)**

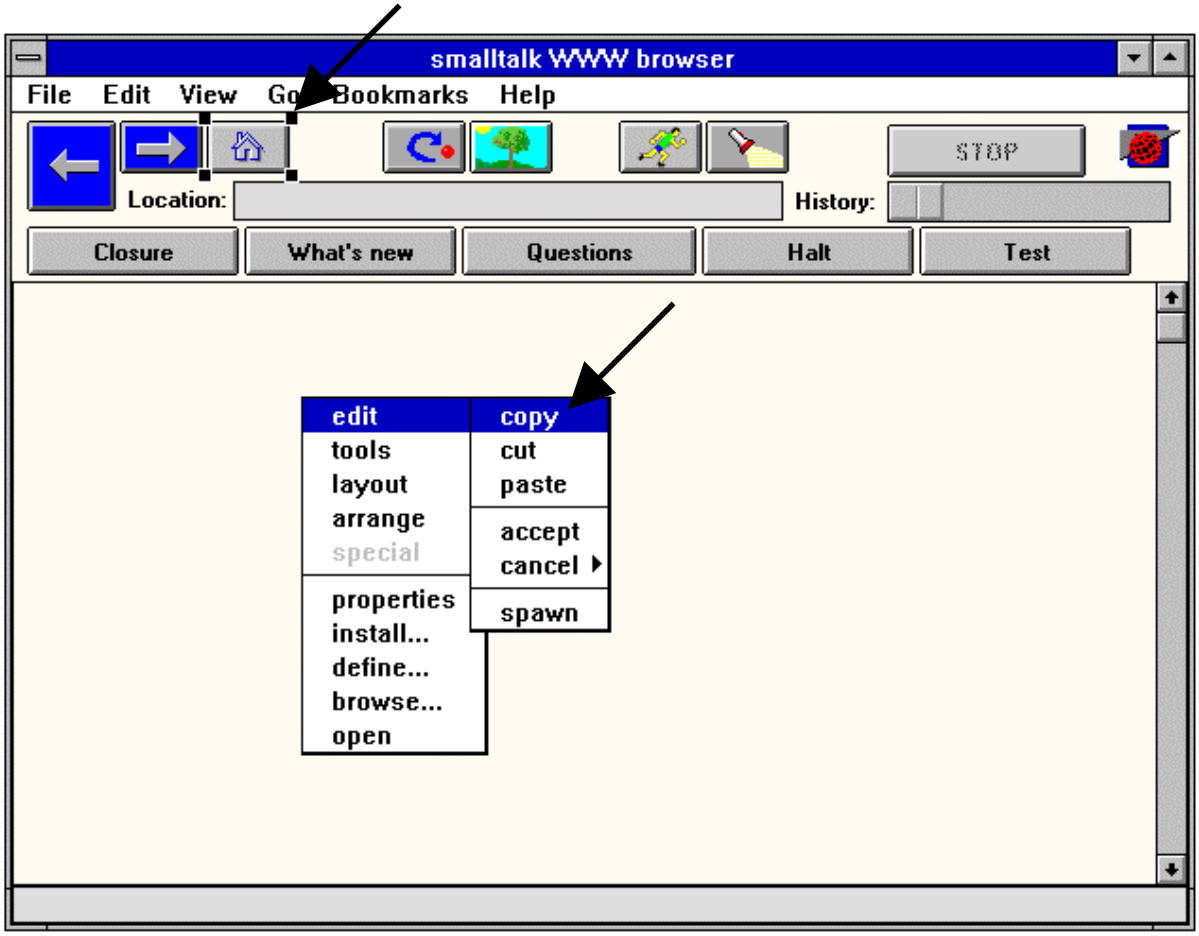


Figure 10

- **Edit the winSpec of the existing application and paste the items. (figure 11)**

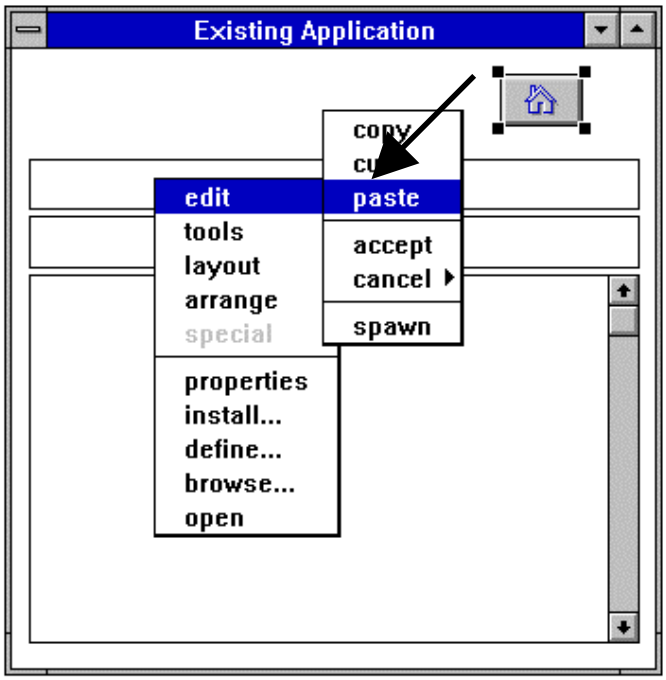


Figure 11

- **Install the existing application on Class: HtmlBrowserApplicationModel.**
  (Since this is a subclass of ApplicationModel the existing application will continue to work properly.) **(figure 12)**

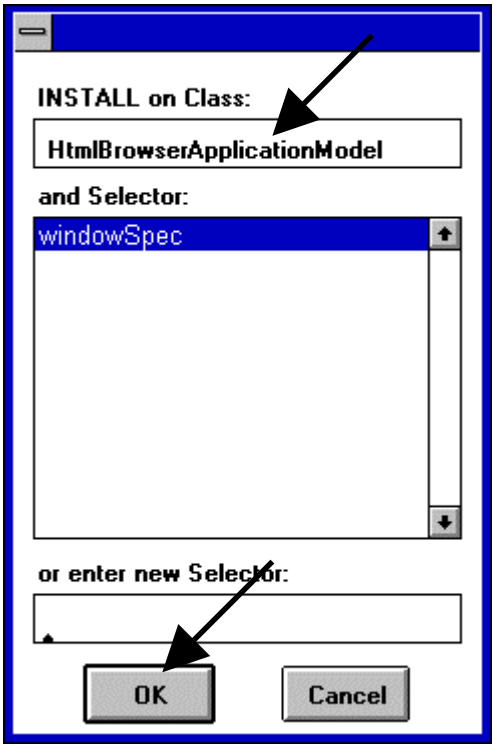


Figure 12

That's all it takes to reuse parts of the html-browser in other applications.

This ease of reuse of user interfaces is unprecedented in other programming environments and if one realize that this reuse can be implemented as easy on the fly, in a dynamic way, then the possible applications are abundant and incredible things become possible in an internet environment where smalltalk code is exchanged. If this code is dynamically generated at the server side in a smalltalk script environment [iii] then things become possible where current WEB users can only dream of.

HTML is the MS-DOS of the web.
Instead of using a static datastructure as HTML one should use something that can execute itself as PostScript does.

Allan Kay, Apple Computer, Inc., USA
in his keynote Speech at the third international WWW conference Darmstadt '95

7. Reusing the framework

The most ideal situation would be that the net-object (the set of html- and other servers , name servers, agents, search engines etc...) sends us an object that can execute itself.

Then we send the message **executeYourself** to the object after it is embedded in an environment (framework) that allows it to see where it is and how it's playground look's alike.

But until now we receive html-documents from servers that we lookup from different sources, so back to reality.

The framework classes are grouped in two main categories

- HTML-Doc
  Contains the classes originally designed by Mark Plas to implement a html editor.
- HTML-Browser
  Contains the classes that form the browser.

Classes for processing gif images and internet access resides in separate categories:

- Graphics-GIF Reading
- OS-Sockets

7.1 Scanning and Parsing

HTML is a markup language for hypertext which is understood by all WWW clients.

At this moment HTML 3.0 is the most recent specification.
In this specification features such as tables, figures, mathematical equations, stylesheets, time dependent actions are in development.

Since the specification of this language seems to change almost daily it would make sense to use the scanning and parsing code that is made available by the W3 consortium at Cern.

The CERN World-Wide Web Library of Common Code[iv] is a general code that can be used to build clients and servers. It contains code for accessing HTTP, FTP, Gopher, News, WAIS, Telnet servers, and the local file system. Furthermore it provides modules for parsing, managing and presenting hypertext objects to the user and a wide spectra of generic programming utilities.

In this implementation however scanning and parsing is done by smalltalk classes from the HTML-Doc category.

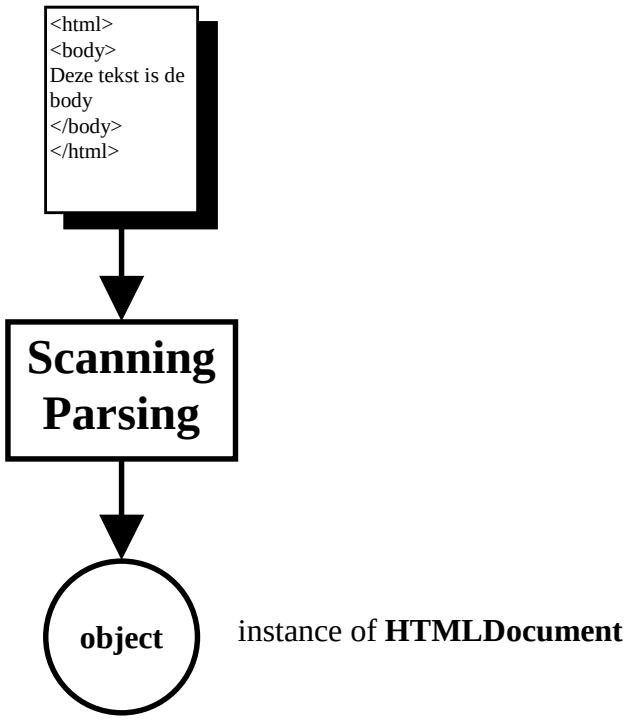


Figure 13

The result of this process is an instance of **HTMLDocument**. (figure 13)
This object contains a hierarchic structure of the hypertext (in instance variable body).

HTMLScanner reads the text file and passes the correct parts of it to

HTMLParser who build the hierarchic tree structure of the document.

The elements of the tree structure can be of three different types :

1. String.
2. TagWithBody. tagged information with a content e.g. bold
3. TagWithoutBody. tagged information without a content e.g. <brk>

The structure of the html text:

<html>
<head>
<title>Test</title>
<link rev="made"href="mailto:lstoops@vub.ac.be">
</head>
<body>

Examples: bold   italic italic and bold

</body>
</html>

Will be displayed on the screen as : Examples: **bold** *italic*, ***italic and bold***
and is internal represented asin figure 14.

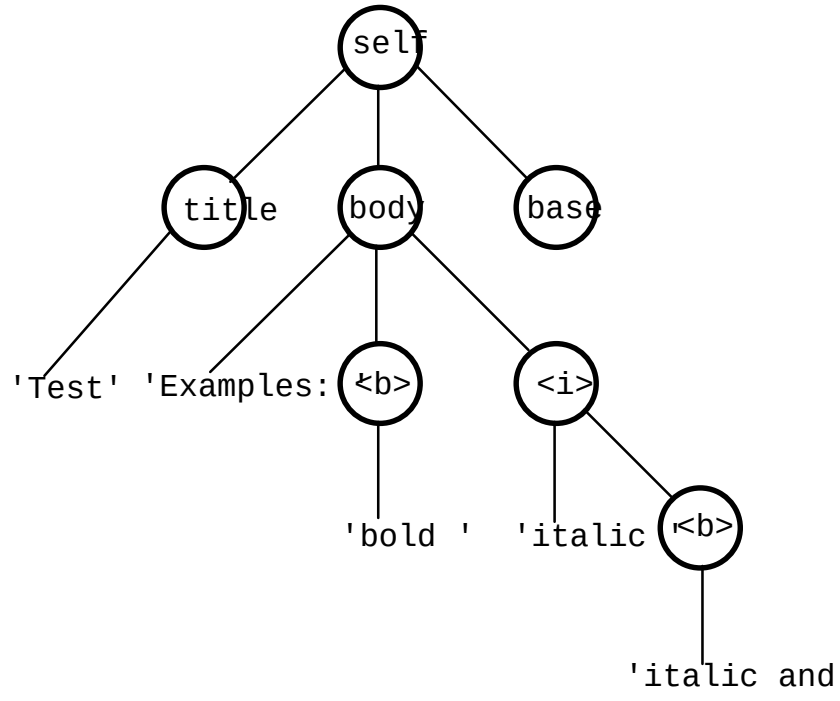


Figure 14

In this tree the node <i> means a TagWithBody where the contents of the body contains the descendants of the node.

7.2 Visualizing the document

The visualisation process converts the tree structure to a list of displayable items used by the htmlView.

The different tags <h1>, <b>, <i> ... needs different methods to render themselves therefore it would have been easier if the different tags corresponded with different classes so all tagclasses could get their own render method. Future applications may also take advantage of having different classes so it would be nice if one could use the CERN World-Wide Web Library of Common Code to scan and parse the html-document and then convert the resulting hierarchic C structure to a tree with the corresponding smalltalk classes.

To allow each item in our parser to render itself all tags have an instance variable (renderBlock) containing a block with the actions needed to render the item.

This block is assigned during the execution of the instance method **setupTags** of **HTMLDocumentParser**.

The part in this method that assigns the H1 tag block look's like this:

self addTag: (TagWithBody newTag: 'H1' terminators: (Set with: '/H1') attributes: #() asOrderedCollection allowedInBody: headingBody renderBlock: [:aTag| TextComposer new setAllHeader1: (aTag renderBody)]).

Most of the block messages uses **TextComposer** to prepare the visual item.

e.g. the **TextComposer** method **setAllHeader1:** that sends **setAllHeader:** to self**.**

It is very important to give text all his attributes and style **before** converting the text to ComposedText or to HtmlComposedText.

If not the width and height method of ComposedText gives not the correct answer!

see **setAllHeader:**

The visualisation of a URL is triggered by a change in the location input field. (figure 15)

Figure 15

The contents of this field can be changed by :

1. Direct keyboard input in the field Location.
2. via menu File OpenLocation.
3. via menu File Open File.
4. via menu Go Home.
5. via menu Go Back.
6. via menu Go Forward.
7. via button Home.
8. via button Back.
9. via button Forward.
10. via the mouse button menu Back.
11. via the mouse button Open Location.
12. via the slider.

Since during the **initialize** method the valueholder location is send the message:

self location onChangeSend: #locationChanged to: self.

after a change in location the method **locationChanged** is launched.

After parsing the contents of location via **URL** and his subclasses **FileUrl** and **HttpUrl** it basically sends a **showHtmlDocument** message to htmlModel (an instance of **HtmlModel**).

This method sends the new added **HtmlDocument** method **render: actualPath imagesStatus: imagesOn** to htmlDocument (an instance of **HtmlDocument** in **HtmlModel**). In this method the displayable items are rendered and a list of them is returned.
The parameter actualPath is needed to form the absolute path of the hyperlinks.

The parameter imageStatus indicates if images are needed or not.

Each tree element that is not a string responds to the message **render** (instance method in TagWithoutBody, inherited by TagWithBody) by returning a displayable object.

The returned list of displayable items is then used by **HtmlView** to build the main window.

Since a standard **ComposedTextView** can only hold one linespacing and baseline it is impossible to show text in different fonts in the same window, moreover the view has to show also hyperlinks and images so the use of a custom view was necessary.

HtmlView implements it's own **displayOn:** method that asks the items of the list to display themselves.

The list consists of objects that are all subclasses of the abstract superclass **HtmlViewItem** (figure 16).

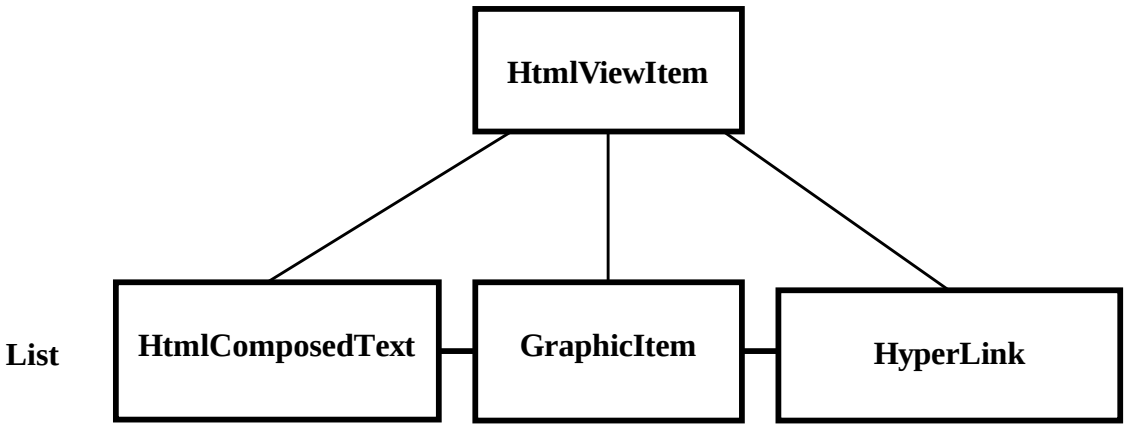


Figure 16

All items must override the **displayOn: gc** method (self subclassResponsibility) of **HtmlViewItem**.

The **HtmlView** method **displayOn: gc** asks all items to display themselves by sending them the message **displayOn:** gc

position: indent @ ceiling
lastHeight: lastHeight
model: self model.

This method is implemented in the abstract superclass **HtmlViewItem** where the message **displayOn: gc** is send "back" to the items.

7.2.1 GIF Images

Class **ImageReader** is an abstract super class representing a reader of images stored in external formats such as Windows BMP, GIF.

Subclasses implement readers for specific formats.
Two subclasses are implemented, the standard bmp reader and a gif reader
BMPImageReader
GIFImageReader ()[v]

Depending on the file extension (*.bmp, *.gif) the class **ImageReader** checks it subclasses to find out if the imagetype is implemented and then send the appropriate messages to the class concerned.

A new method **imageFromFile: aFilename** was added to the class **HTMLDocument** so that the class could read it's images.

The image returned from the method respond to the standard messages **displayOn: gc , width and height.**

7.2.2 HyperLinks

The instance variable hyperLinks in **HtmlModel** contains a list of associations of rectangles and url's.
As shown in the **displayOn: gc** method of **HyperLink**, for each word of a hyperlink text, a dictionary is added to the list. (figure 17)

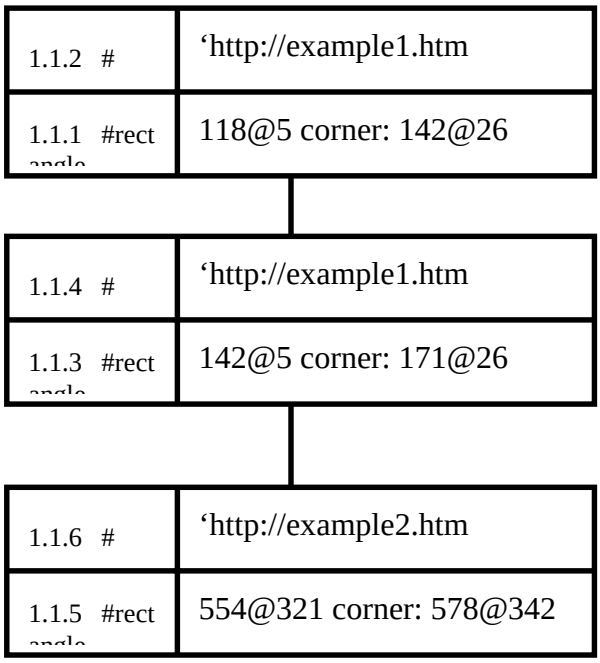


Figure 17

When the left mouse button is pressed (red button) the message **redButtonActivity** is send to **HtmlController**. In this method the list is traversed to find out if the actual position point is contained in one of the rectangles, if so location gets the corresponding url and the file is automatically fetched and displayed.

8. Accessing the internet

A class method **getFile: fname from: host** of **HttpClient** implements the TCP/IP access and returns a string with the contents of the filename 'fname' from the host 'host' using the HTTP protocol.

When the location input field is changed a **locationChanged** message is send to the **HtmlApplicationModel** and if the url appears to be an http call then the message **getHttpFile: hostPath at: hostPort** is send to htmlModel which gives htmlDocument it's new value.

9. The Classes

This is an overview in alphabetically order of the classes and instance variables that makes up the html browser framework.

Class: **FileUrl**

Superclass: URL
Category: HTML-Browser
Instance variables: msPath

- msPath <String> with the filename in ms-dos format

Class: **GraphicItem**

Superclass: HtmlViewItem
Category: HTML-Browser
Instance variables: image source noImagesIcon

- image <Image> the actual image
- source <String> relative name e.g. drawing.gif
- position <Point> left top point to start the display
- lastHeight <Number> lowest point of the actual row.

Class: **HtmlBrowserApplicationModel**

Superclass: ApplicationModel
Category: HTML-Browser
Instance variables: htmlModel commentString location htmlView defaultBackground htmlHistory historyPosition historyRange

- htmlModel <Model>
- commentString <ValueHolder on String> string with comments at bottom of the window
- location < ValueHolder on a URL> active url
- htmlView <HtmlView>.
- htmlHistory <HtmlHistory>
- historyPosition < ValueHolder on number>
- historyRange <RangeAdaptor> for dynamic slider range
- defaultBackground <Boolean>

Class: **HtmlComposedText**

Superclass: HtmlViewItem
Category: HTML-Browser
Instance variables: composedText

- composedText <ComposedText>

Class: **HtmlController**

Superclass: ControllerWithMenu
Category: HTML-Browser

Controls the mouse actions when the mousepointer is in the html view.

Class: **HTMLDocument**

Superclass: Object
Category: HTML-Doc
Instance variables: title body base

Class: **HTMLDocumentParser**

Superclass: Object
Category: HTML-Doc
Instance variables: tags attributes scanner buildTreePos ampersandTokens title base preformatted

---

Class: **HTMLDocumentScanner**

Superclass: Object
Category: HTML-Doc
Instance variables: source hereChar token tokenType tokenTable

---

Class: **HtmlHistory**

Superclass: Object
Category: HTML-Browser
Instance variables: history positionChanged historyAppended

- history <list of Dictionary's with htmlDocuments and urls>
- positionChanged <Boolean>
- historyAppended <Boolean>

Class: **HtmlModel**

Superclass: Model
Category: HTML-Browser
Instance variables: htmlDocument hyperLinks htmlText imagesOn applicationModel htmlSource

- htmlDocument <HTMLDocument> active document.
- hyperLinks <list of dictionaries> list of current hyperlinks
- imagesOn <Boolean>
- htmlText <List of displayable objects>

- applicationModel<HtmlBrowserApplicationModel>
- htmlSource <String> active document in source format.

Class: **HtmlView**

Superclass: AutoScrollingView
Category: HTML-Browser
Instance variables: windowHeight viewColor

implements the view of a html document in the view window

- windowHeight <Number> actual window height
- viewColor <Color> backgroundcolor

Class: **HttpClient**

Superclass: Object
Category: HTML-Browser

implements the http protocol to access to files

Class: **HttpUrl**

Superclass: URL
Category: HTML-Browser
Instance variables: hostPort hostPath search

- hostPort <String> host name
- hostPath <String> pathname + filename
- search <String> not yet used

Class: **HyperLink**

Superclass: HtmlViewItem
Category: HTML-Browser
Instance variables: body href fullPath

- body <text list> hyperlink text in blue
- href <String> relative adress of link
- fullPath <String> absolute adress of link

Class: **ScreenControl**

Superclass: HtmlViewItem
Category: HTML-Browser
Instance variables: controlCode

- controlCode <Symbol> #cr is the only code used at this moment

---

Class: **TagWithBody**

Superclass: TagWithoutBody
Category: HTML-Doc
Instance variables: terminators body

- tagName <String> contains the name of the tag
- terminators <Set> contains the names of the tags that can announce the end of this tag
- attributes <OrderedCollection>
- contains the list of attributes that this tag can have.
- allowedInBody <Set> contains the names of the tags that are allowed in the body of this tag.
  Text is always allowed. An empty set means that only text is allowed.
- renderBlock <Block> a block to render the information.

Class: **TagWithoutBody**

Superclass: Object
Category: HTML-Doc
Instance variables: tagName attributes instVars allowedInBody renderBlock

See the comment from TagWithBody

Class: **TextComposer**

Superclass: Object
Category: HTML-Browser

This class implements the methods to give emphasis at the different html text.
It seems to be very important that the emphasis is given to text before converting the text in composedText (or htmlComposedText) since if it is done the other way around the composedText fails to calculate the width and height correct !

Class: **URL**

Superclass: Object
Category: HTML-Browser
Instance variables: urlString scheme rest

- urlString <String> total url
- scheme <String> 'file' or 'http'
- rest <String> rest of total url

10. Related work

During the development of the browser I got stuck twice. Once with the **ComposedText height** and **width** methods (see warning at page 19) and a second time when I was trying to adapt the length of the active window to the height of it's contents (solved by overriding the inherited **preferredBounds** method in **HtmlView**). In both cases I was helped out, via usenet, by Hans-martin Mosner, a Smalltalk programmer at George Heeg objektorientierde Systeme in Dortmund.

At that time he was working on a html-browser/editor now called Documenta-WWW[vi] in Visual Works 2.0.
Now, August '95, the first version of the Documenta-WWW browser/editor Documenta-WWW 1.0 beta1 is released but at this moment it 's only available for Georg Heeg employees.

Having experienced the programming skills of the author it must be a powerful tool that certainly deserves further exploration.

Documenta-WWW is a WWW browser with integrated editor that makes HTML editing much
easier and safer. Among its features are:

- **HTML 3.0** compliance (figures, tables, math)
- Full-blown SGML system with robust error handling
- WYSIWYG editing (using standard rendering methods of the classic browsers)
- Web maintainer tools
  planned are:
  - Link checker
  - Document Structure View
  - Directory Structure View
  - Imagemap Editor
  - Navigational Pages Editor

- Extendibility (it's written in Smalltalk Visual Works 2.0)

The following description of the menu bar gives us an idea what to expect.

File

New Window
Open a new window.
Reload

Load, parse and display the current URL again.

Save

Store the current page in the file corresponding to the current URL.

Save As...

Store the current page in a named file.

Mail To...

Mail the current page to someone.

Print

Produce a hardcopy of the current page.

Quit

Close this window.

Edit

Copy

Copy selected text (not yet implemented; use the pop-up menu or Copy key instead.)

Cut

Cut selected text (not yet implemented; use the pop-up menu or Cut key instead.)

Paste

Paste copied text over selection (not yet implemented; use the pop-up menu or Paste key instead.)

Go

Back

Go back to the previous page.

Forward

Go forward to the next page (only available if Back has been executed before.)

Home

Go to the home page (configurable in the settings dialog.)

Open URL...

Open a specified URL.

Open from Clipboard

Open a URL that's in the clipboard (for example, copied from a mail or news reader.)

History

Open a window listing the pages visited in this browser.

Options

Show Toolbar

Switch toolbar display on/off.

Show URL

Switch URL display on/off.

Show Status Line

Switch Status Line display on/off.

View Source

Switch between formatted and source display.

Settings...

Open a Settings dialog.

Utils

Reformat

Parse and display the current page, for example when the window width has changed and tables should accommodate to the new width.

Flush URL Cache

Remove all cached URL entries.

Open Structure View

Open a window showing the structure of the HTML document.

Open Document Tree

Open a window showing the directory structure of the directory containing the current URL.

Open Tag Inserter

Open a palette of tags (obsolete.)

Style

EM

Insert the <EM> tag (typically rendered as italics)

STRONG

Insert the <STRONG> tag (typically rendered as bold)

CODE

Insert the <CODE> tag (typically rendered as fixed width)

SAMP

Insert the <SAMP> tag (typically rendered as fixed width)

KBD

Insert the <KBD> tag (typically rendered as bold fixed width)

Para

(to be done)

Heading

H1 .. H6

Insert a heading of the appropriate level.

List

OL, UL

Insert an Ordered List or Unordered List, respectively.

LI

Insert a List Item.

DL
Insert a Definition List.
DT, DD
Insert a Defined Term or a Definition, respectively.

Attributes

...
(items are dynamic, according to the currently selected element.)

Help

About...
Shows a dialog with the copyright and version information.
User Guide
Open a window with the user guide for the current version.
Mail Developers
Send mail to the developers.
Error List
Open a list of HTML errors for the current document.

The following options are in the development version only:

Inspect
Inspect the WWW Browser.
Inspect Controller
Inspect the main text controller.
Inspect ComposedText
Inspect the formatted text.
Inspect Element
Inspect the HTML element

11. Conclusion

Frameworks and design patterns brings the job of object-oriented design to higher levels of abstraction but there is still no guarantee to successful reuse of the code.

One of the problems for potential reusers is that it is not always clear what the intention of the designer of the framework is and how objects of different classes are supposed to interact with each other.

Smalltalk code in hypertext[vii] can help. In this WWW approach methods are coded in hypertext where words (classes, messages, temporary variables, symbols etc.) are links to a presentation of the classes and their relations, example code, references to external documents etc.

In this document I tried to explain the structure of the html-browser framework in detail and I suggested ways to reuse the code of it.

But a framework is just a framework and it is left to the end-user to exploit it's intended possibilities or to creatively adapt it to other purposes. (figure 18)

Figure 18 (from Bauherrlichkeit - München : Gabor Benedek)

12. Appendix

This appendix lists most of the methods referenced in the document in order of appearance.

---

class method of HtmlBrowserApplicationModel

open
```
self open: 'file:///c|/luk/vub/html/luk/welcome.htm'
```

---

class method of HtmlBrowserApplicationModel

open: aUrl
```
super open source location value: aUrl
```

---

instance method of TextComposer

setAllHeader1: textList

```
"set the text in textList in 70 pixels"


^self setAllHeader: textList pixelSize: 70
```

---

instance method of TextComposer

setAllHeader: textList pixelSize: pixelSize
```
"set the text in textList at pixelSize."

| ctl ca style ct |
ctl := List new.
textList
    do:
        [:text |
        ca := CharacterAttributes newWithDefaultAttributes.
        ca setDefaultQuery: (TextAttributes default fontAt: nil).
        ca at: #header put: [:fontDesc | fontDesc pixelSize: pixelSize].
        style := TextAttributes styleNamed: #systemDefault; characterAttributes: ca.
```

```
        style gridForFont: #header withLead: 0.
        ct := HtmlComposedText  withText: (Text string: text emphasis: #(#header #,
#bold))
                style: style.
        ctl add: ct].
    ctl add: (ScreenControl new: #cr).
    ^ctl
```

---

instance method of HtmlBrowserApplicationModel

initialize

```
    "announce how to send the change notification and initialize instance variables"

    super initialize.
    self location onChangeSend: #locationChanged to: self.
    htmlHistory := HtmlHistory new: self historyPosition.
    self historyPosition onChangeSend: #historyPositionChanged to: self.
    self historyRange: (RangeAdaptor
        on: historyPosition
        start: 1
        stop: 2
        grid: 1).
    self htmlModel: HtmlModel new.
    self htmlModel applicationModel: self.
    self htmlView: (HtmlView new model: htmlModel).
    self htmlModel imagesOn: true.
    defaultBackground := true.
    self htmlView background: (ColorValue
        red: 0.75
        green: 0.75
        blue: 0.75).
```

---

instance method of HtmlBrowserApplicationModel

postBuildWith: aBuilder

```
    "assign a WWW icon to the collapsed window.  This can not be done in the initialize
     method since the builder is not available at that moment"

    | win image mask icon |
    win := aBuilder window.
    image := self class winicon.
    mask := image asRetainedMedium.
    icon := Icon image: mask.
    win icon: icon
```

---

instance method of HtmlBrowserApplicationModel

locationChanged

```
"message is send to self if location was changed
 this triggers the processing of the new url"

| url fileUrl httpUrl |
location value = ''
    ifFalse:
        [url := URL new.
        url value: location value.
        url isFile
            ifTrue:
                [fileUrl := FileUrl new.
                fileUrl value: location value.
                htmlModel openLocalFile: fileUrl msPath asFilename]
            ifFalse: [url isHttp
                    ifTrue:
                        [httpUrl := HttpUrl new.
                        httpUrl value: location value.
                        htmlModel getHttpFile: httpUrl hostPath at: httpUrl hostPort]].
        htmlHistory positionChanged
            ifFalse:
                [htmlHistory history add: location value.
                htmlHistory historyAppended: true.
                htmlHistory history size = 1
                    ifTrue: [self historyRange rangeStop: 2]
                    ifFalse: [self historyRange rangeStop: htmlHistory history size].
                historyPosition value: historyPosition value + 1].
        htmlHistory positionChanged: false.
        htmlModel showHtmlDocument]
```

---

instance method of HtmlModel

showHtmlDocument

```
"render the htmlDocument and show it in the window"

| url wrapper  |
url := URL new.
url value: self applicationModel location value.
self htmlText: (self htmlDocument render: url path imagesStatus: self imagesOn).
```

```
    wrapper := self applicationModel htmlView container.
    wrapper isNil ifFalse: [wrapper scroll: wrapper scrollOffset].  "show top of window"
    self changed
```

---

instance method of HTMLDocument

render: actualPath imagesStatus: imagesOn

```
    "answer a list of items and make sure that all text is presented as HtmlComposedText and
    that all Graphic Items get their image"

    | itemList tc list ctl url fUrl |
    itemList := List new.
    self body do: [:each | each isString
            ifTrue:
                ["pure string without tag's"
                tc := TextComposer new.
                list := tc wordList: each.
                list do: [:word | itemList add: (HtmlComposedText withText: word)].
                itemList add: (HtmlComposedText withText: ' ')]
            ifFalse: [itemList addAll: each render]].
    ctl := List new.
    itemList do: [:item | item class = Text
            ifTrue: [ctl add: (HtmlComposedText withText: item)]
            ifFalse: [item class = GraphicItem
                    ifTrue: [imagesOn
                            ifTrue:
                                [url := URL new value: actualPath , item source.
                                url isFile
                                    ifTrue:
                                        [fUrl := FileUrl new.
                                        fUrl value: url value.
                                        item image: (self imageFromFile: fUrl msPath).
                                        ctl add: item]
                                    ifFalse: ["(image via http call)"]]
                            ifFalse: [ctl add: (GraphicItem new image:
HtmlBrowserApplicationModel noImages)]]
                    ifFalse:
                        [item class = HyperLink ifTrue: [item fullPath: actualPath , item href].
                        ctl add: item]]].
    ^ctl
```

---

instance method of tagWithoutBody

render
```
"execute the renderBlock on myself"

^renderBlock value: self
```

---

instance method of HtmlView

displayOn: gc
```
"ask the list of items to display themself"

| itemList indent ceiling lastHeight |
self container container insideColor: viewColor.
self model hyperLinks: List new.
indent := 5.
ceiling := 5.
lastHeight := 0.
itemList := self model htmlText.
itemList
    do:
        [:item |
        item
            displayOn: gc
            position: indent @ ceiling
            lastHeight: lastHeight
            model: self model.
        indent := item indent.
        ceiling := item ceiling.
        lastHeight := item lastHeight].
windowHeight := ceiling + lastHeight + 5 max: 360.
self container scroll: 0 @ 0
```

---

instance method of HtmlViewItem

displayOn: gc position: aPosition lastHeight: aLastHeight model: aModel
```
"display yourself in the view at aPosition taking in account the lastHeight."

indent := aPosition x.
ceiling := aPosition y.
```

```
    lastHeight := aLastHeight.
    model := aModel.
    self width + indent > maxWidth ifTrue: [self nextLine].
    self displayOn: gc.
    lastHeight := self height max: lastHeight.
    indent := indent + self width
```

---

instance method of HTMLDocument

imageFromFile: aFilename

```
    "answer the filecontent as image"

    | reader image |
    reader := ImageReader fromFile: aFilename.
    image := reader image.
    ^image
```

---

instance method of HyperLink

displayOn: gc

```
    "display yourself in the view
    for each word append the hyperLinks list with your view rectangle and fullPath"


    | textList htmlText textWidth dict |
    textList := self body.
    textList
        do:
            [:hrefText |
            htmlText := ComposedText withText: hrefText.
            textWidth := htmlText width.
            textWidth + self indent > maxWidth ifTrue: [self nextLine].
            htmlText displayOn: gc at: self position.
            dict := Dictionary new.
            dict at: #rectangle put: (self position extent: htmlText width @ htmlText height).
            dict at: #link put: self fullPath.
            model hyperLinks add: dict.
            self lastHeight: (htmlText height max: self lastHeight).
            self indent: self indent + textWidth].
```

---

instance method of HtmlController

redButtonActivity
"load the document under the hyperlink if the mouse pointer points to one"

```
| point cursorOnLink list fullPath |
point := self sensor cursorPoint.
cursorOnLink := false.
list := self model hyperLinks.
list do: [:dict | ((dict at: #rectangle)
        containsPoint: point)
        ifTrue:
            [fullPath := dict at: #link.
            cursorOnLink := true]].
cursorOnLink ifTrue: [self model applicationModel location value: fullPath]
```

---

class method of HttpClient

getFile: fname from: host
"this method gets the file 'fname' from the host 'host' using the HTTP protocol"

```
| port socket connection stream reply |
port := 80.

"Create a socket on the given host and port."
socket := SocketAccessor newTCPclientToHost: host port: port.

"Open a two-way connection on the socket."
connection := ExternalConnection new.
connection
    input: socket;
    output: socket.

"Open a stream on the socket connection."
stream := connection readAppendStream.
stream lineEndTransparent.

"Send a message to the server, then get the reply."

stream nextPutAll: 'GET ' ,fname; cr;  nextPut: Character lf ;  commit.
```

```
    (Delay forMilliseconds: 200) wait.
    reply := stream contents.

    "Close the stream (which closes the socket)."
    stream close.
    ^reply
```

---

instance method of HmlModel

getHttpFile: hostPath at: hostPort

```
    "get the file hostPath using tcp/ip"

    | stream |
    self htmlSource: (HttpClient getFile: hostPath from: hostPort).
    stream := ReadStream on: self htmlSource.
    self htmlDocument: (HTMLDocumentParser parser: stream).
    stream close.
    applicationModel builder window label: self htmlDocument title
```

---

instance method of HmlView

preferredBounds

```
    "overwrite the inherited prefferedBounds method to supply the actual prefferredBounds"

    ^0 @ 0 extent: 600 @ windowHeight
```

---

13. Index

I

L

M

O

P

Q

R

S

T

U

V

W

Y

14. Literature

Kroemker D., *Proceedings of the Third International World-Wide Web Conference, Computer Networks and ISDN systems* Volume 27 (1995) Number 6 April 1995.
http://www.elsevier.nl/ password: els4www3.

Holzapfel Roland, *Poster Proceedings Third International World-Wide Web Conference Darmstadt '95.*
http://www.igd.fhg.de/www95.html

Hardin Joseph, *Tutorial Notes Third International World-Wide Web Conference Darmstadt '95.*
http://www.igd.fhg.de/www95.html

Gamma Erich, Design Patterns: *Abstraction and Reuse of Object-Oriented Design*.
Department of Computer Science. University of Illinois at Urbana-Champaign.

Johnson Ralph E., *Designing Reusable Classes*.
Department of Computer Science. University of Illinois at Urbana-Champaign. August 26, 1991.

Oberon/F *From Applications to Applications Frameworks* Introduction part 1.2
Oberon microsystems,Inc. Base, Switzerland.

Leveraging Object-Oriented Frameworks, A technology primer.
http://www.taligent.com/leveraging-oofw.html
Taligent, Inc., 1994

Deutsch Peter L. *Design reuse and frameworks in the smalltalk-80 system*
ParcPlace Systems, Inc.

VisualWorks Tutorial,
ParcPlace Systems, Inc.

VisualWorks Cookbook,
ParcPlace Systems, Inc.

VisualWorks User's Guide,
ParcPlace Systems, Inc.

Goldberg Adele *Smalltalk-80 The language*
Addison-Wesley Publishing Company

15. References

[i] Johnson, Ralph E., *Designing Reuseable Classes*, Journal of Object-Oriented Programming August 26, 1991.

[ii] Catledge Lara D. Characterizing browsing strategies in the World-Wide Web Computer Networks and ISDN Systems 27 (1995) 1065-1073.

[iii] Plas Mark, Thesis about scripting in a html server in Smalltal*k* . Thesis academic year 1994-1995 Brussels Free University.

[iv] http://www.w3.org/hypertext/WWW/Library/Status.html

example heading

```
/*      General SGML Parser code    SGML.c
**      ========================
**
** This module implements an HTStream object. To parse an
** SGML file, create this object which is a parser. The object
** is (currently) created by being passed a DTD structure,
** and a target HTStructured object at which to throw the parsed stuff.
**
**  6 Feb 93   Binary searches used. Interface modified.
**  8 Jul 94  FM  Insulate free() from _free structure element.
*/
```


[v] http://www.heeg.de/pub/hmm-goodies.

[vi] http://www.heeg.de/~hmm/Documenta-WWW/1.0beta1.

[vii] http://www.heeg.de/~hmm/objman/HyperLiterate.html