AUIS and the World Wide Web

An update on current work

Tom Neuendorffer
tpn@galt.com

Introduction

Last year I presented a draft specification for what I intended to do. Because of a number of factors, this work has gone much slower that I expected. The following is a subset of that document, with the addition of updates on the current status of this work, notes on it's implementation, and a new conclusion on future directions.

Overview

The World Wide Web was originally conceived of at CERN. To quote their project document:

> *The WorldWideWeb (W3) is the universe of network-accessible information, an embodiment of human knowledge. It is an initiative started at CERN , now with many participants. It has a body of software, and a set of protocols and conventions. W3 uses hypertext and multimedia techniques to make the web easy for anyone to roam, browse, and contribute to.*

A Web browser has the ability to view data from a wide variety of sources, including ftp, gopher, and http servers. By combining this functionality with AUIS's ability to create and display data, a powerful information system can be created. The Web community has been working on an expanded definition of their Mark-up language (called HTML+). Because it already supports a variety of embedded objects, AUIS has the potential to be one of the first Web browsers to fully implement this new standard.

The World Wide Web has become the medium of choice for the dissemination of information on the Internet. The goal of this project is to integrate the AUIS and Web environments as seamlessly as possible, in order to facilitate transference of information both to and from AUIS users. In the process, AUIS should become a first class environment for both accessing and creating Web documents, and an example for how the Web can be integrated into a user environment. The result should be an increase in public interest in and awareness of AUIS.

An initial HTML inset, supporting both display and editing of html documents, has been implemented by Nick Williams at City University in London. An updated HTML editor is also in progress at IBM Rochester. Building on top of their work and the W3 library from Cern, I have implemented a prototype Web browser in ATK.

Object Descriptions

Web Inset

The Web inset will be sub-class of the HTML inset. When the webview receives it's first _FullUpdate request , it would make a call to the webcom object to bring up the appropriate file, firing off an external viewer when necessary. By tying the load to the FullUpdate routine, a document that contains a number of Web insets will only access them when necessary to view them.

When a Web inset is told to read a new URL (either via a click on an embedded anchor, an Open URL or Open Local command, or a procedure call), the following procedure is followed:

editing (if any) of the current Web object is suspended,
a new webcom object is created,
a timer loop (utilizing im_Interact) initiated that will
    check on the progress of the webcom
    notify the user of that progress
    display an animation indicating web activity
    check for any user action to indicate that the web request should be canceled

If the read is successful, then
    if the object is of type text, it will replace the current text.
    if it of a different type, webcom_View will be called to spawn an external viewer
If the read was unsuccessful or not of type text, then the last URL will continue to be viewed.

Current plans include support for an authoring mode in Web insets, where links will not be warped to , and normally invisible text and attributes will be visible. Details on this remain to be worked out.

Current plans also call for requests for visiting a web document coming from outside of the web document itself to be directed to a single window (probably running in a separate process). In addition, one should have the capability of creating other web windows that won't be affected by external requests.

Status:

The web inset is implemented.

No animated icon is in place, however reports of how much of the object has been read are displayed in the message line, and a cancel read button is implemented.

Added support for authoring still needs to be worked out.

Support for all web requests to go to a single window is implemented. The AUIS web browser is now assumed to be running in a separate process and a small 'weblnch' class has been implemented so that any other AUIS application can request a URL to be imported and displayed.

Weblink Inset

The weblink will be a subclass of the link inset that would contain a URL and would , when clicked , use webcom calls to bring up the appropriate file, using Web Inset when possible, or firing off an external viewer when necessary.

Status:

This is now implemented using the 'weblnch' class mentioned above. It also contains class procedures for extending and text based object, allowing the user to select any URL and request the display of it in the web window. If no web window exists , one is created.

Webcom

Webcom objects will have a lot of functionality , but will be small in terms of data (mostly a pointer to a web object , a file pointer, a file name, and a URL name) . We will be able to reclaim the web object, and associated buffer at any time via the Cull method, while the information to reload if necessary will remain in the webcom. Webcoms will probably never be destroyed. By using the parent/child pointers , the webcom objects will provide their own history that can be used for back/forward navigation. They will also be linked in order of creation, which will be used to support a global history function (window history, as supplied by Mosaic , make less sense in an ATK model where buffers may show up in any window).

A WEBCOM object may point to a web object, it's containing buffer, a file pointer, and or a file (which may be in /tmp). Each Web object can be related to, at most , one Webcom object, and each webcom object can have no more than one web object, but either can exist without the other. To do anything, a Web object needs an associated webcom, but one is not automatically created when a web object is created.

The basic webcom can be created and used without a web object to access the file for an inline image, or to trigger the display of a Web document in an external process. Also the Webcom will be able to bring up external viewers on non-HTML data, which is another case where there will be no Web object.

Weblink objects will contain a pointer to one webcom object. There will be no back pointer. Multiple weblinks can point to the same webcom.

Webcoms are subclasses of observable, however both web and weblink objects will want to save enough information in their datasteams to reproduce them. Thus they will include methods to read and write the critical information to a stream.

Status:

Once the decision had been made to have most of the interaction with the web view come from outside processes, the described level of webcom functionality has been somewhat deemphasized, while more work has gone the web browser as an application.

User Interface

Unified Operations and EZ support

Description

The behavior of the following functions was designed to provide consistency of action over a range of insets, tailoring the action produced to the inset involved. Naturally, all of the following menu commands will also be callable via custom keystroke commands.

Menu Command: Visit WEB

Objects supported:

Frame level command, available to all ez clients.

Behavior:

If no web window exists, create one, visiting the default home page (setable in preferences). Current plans call for all subsequent requests for visiting a web document to be directed to a single window (probably running in a separate process), though one should have the capability of creating other web windows that won't be affected by external requests.

Menu Command: Visit URL

Objects supported:

Frame level command, available to all ez clients.

Behavior:

Like Visit Web, except that it prompts for an initial URL

Menu Command: Warp to URL

Objects supported:

Any text based client (messages, typescript, text,etc.)

Behavior:

Like Visit Web, except that it will look for a URL in the currently selected text (or surrounding a zero length 'dot') and bring up that document in the web window.

Menu Command: Visit URL in Cutbuffer

Objects supported:

Frame level command, available to all ez clients.

Behavior:

Like Warp to URL, except that it will read the contents of the cut buffer to get a URL to display, thus providing support for non-text based clients that can write plain text to the cut buffer. These will include non-ATK based clients, like xterm. In this case, one would select text in the xterm, move to any ATK based client, and execute this command.

Menu Command: Convert URL's to Links

Objects supported:

Any writable text based client (sendmessage, text, etc.)

Behavior:

Automatically converts URL's in selected text to weblink objects, except in html text sub-classes, where they will be converted to anchors instead.

Status :

Warp, Visit URL , and Convert are all supported by proctable entries and class procedures in the weblink class. Visit Web and Visit URL in cutbuffer are functionalities easilly supported by the Visit URL function.

Menu Command: Make URL

Objects supported:

Any object that has a representation that can be viewed via the web.

Behavior:

Still to be worked out. The goal is to install the document(s) being edited into a hypertext tree somewhere (possibly modifying embedded links in the process) . There are several possible approaches. One could create a map where , at some level, a local file hierarchy could be mapped to a HTTP servers document hierarchy. Tools could be written to inform ATK about any number of such maps and to help maintain the maps themselves. Nick Williams is investigating the idea of having the editor interact with the http server itself to allow authorized users to "publish" documents via the server.

Status:

None

The Web Inset

In addition to the above unified operations, the Web inset will support the following menu commands.

WEB FILE menu

Open URL
prompt for a URL and replace the current buffer contents

Open Local
Use the HTML inset to visit a local file
(Note, now speced so that any file will be brought up in a separate window)

Reload Current
Reread the current document

Refresh Current
Redisplay the current document

Save As...
Writes the current document to a file. Prompts for desired format.

View Src
Visit the file with the standard text inset

Clone Window
Makes a new window on the current document

Status: All currently implemented except reload .

WEB OPTIONS menu

Delay Image Loading (toggle)

Will delay the loading of inline images. Instead a small icon will be displayed instead of plain images and an arrow icon will be displayed instead of images that act as hyperlinks. Clicking on the icon will load the image , clicking on the arrow will go to that link. The *Load Images In Current* menu option will load all images in the current document.

Load Images in current

Load all images in the current document.

Clear Global History

Support for Keeping History and displaying as links.

Status : None supported

Navigate Web menu

Back
Moves to previous document in history (if any)

Forward
Moves to next document in history (if any)

Home Document
Moves to home document

Display History
Display a clickable list of the history of the current session.

Hotlist
Display a clickable list of hotlist items. This may be kept as just another HTML document, rather than a simple list. This would make it easier to annotate and edit, and keep the interface consistent. The difference would be that it would change the document display in a separate window, rather than it's own.

Add Current to Hotlist
Adds the current item to the hotlist.

Internet Starting Points
Pointers to Defined documents

Internet Resources Meta Index
Pointers to Defined documents

Status : All Supported. Hotlist is compatible w/ the one generated by Mosaic. The hotlist is implemented by converting the Mosaic style hotlist to an html document, and then bringing that document up in an overlayed window.

Annotate Web menu

Annotate
Pops up a window for entering a personal annotation to this link.

Edit Annotation
Pops up a window for editing the personal annotation to this link.

Delete annotation
Deletes the personal annotation to this link.

Status: Not Yet Supported

Webview Application layer

The webview will probably have more than one application layer available, depending on if it is a top level object. If it is a top level object (or has it 'TopLevel' attribute set) it will include a series of buttons for some of the common functions of the Web inset. It will also have a graphical icon that will animate when a URL is in the process of being accessed. If it is not a top level object, it's application layer will be the same as htmlview.

Buttons will be provided for the following functions.

Back
Moves to previous document in history (if any)

Forward
Moves to next document in history (if any)

Home
Moves to home document

Reload
Reread the current document

Open---
prompt for a URL and replace the current buffer contents

Save As...
Writes the current document to a file.

Clone...
Makes a new window on the current document

Close Window
Deletes the current window

It will also contain an animation to indicate net activity which can also act as a button to cancel said activity.

Status: No animation, but other features supported

Weblink

In addition to the Unified functions mentioned above,weblink will inherit it's commands from the link object., with the following exception.

Link to URL
This will replace the 'link to file' option of the link inset.

Status : Supported

Current Deficiencies

URLs containing #fields to go to contain no support for moving to that place in the file.

The current structure of the HTML objects makes this very difficult to support (at least I haven't found the magic).

No support for forms

No support for annotations

Dumb about certain Mime types

Port to C++ not completed

Code works to deal with simultaneous reads, but doesn't always do the right thing

Typical multi-threading problems,
i,e. when to kill what.

Needs work on the clean up of temporary files.

Needs work on the reuse of already loaded images.

Needs to be upgraded to latest WWW library.

Accomplishments

Text and images load concurrently, taking advantage of the multi threaded www lib

ftp and gopher supported

Read interruption supported (partially)

History supported

Hotlist supported

Remote control functions supported

The thing is actually usable.

Notes

Use of WWW library was much more difficult than it should have been.

Convoluted structure
No 'Put' support (until recently)
Currently runs as seperate process and writes files to /tmp

Structure of ATK not always ideal,

difficult to preset gif sizes.
difficult to locate labels.

HTML a poor standard.

Possible number of failure modes is enormous.

Multiply html fm (failure modes)* http fm * www lib fm * user fm * atk fm * the number of ATK objects * the number of www lib threads

Possible future work

Table support
Forms
Frames

Conclusion

While producing an AUIS based web-browser has been an interesting exercise, the world has changed in the time it has taken to produce it. It is now hard to conceive how the result can compete in a world where companies like Netscape are investing thousands of man hours in portable browser products that are virtually free. HTML is a moving target, and the latest Netscape announcement includes enhancements such as Hot Java, frames, a scripting language, progressive JPEG, streaming audio and video, HTML editing and security enhancements, as well as support for third party "Inline plug-ins" (i.e. embeddable objects) for PDF, Quicktime movies, Macromedia Director presentations, and other to be named later.

The bottom line, to me, is that the technology we pioneered 10 years ago has finally hit the big time, and that we were the grandfather of networked, portable, text based, embedded object, WYSLRN[1] UI systems. Somebody had to do it, after all. The question is, where do we go from here, specifically as far as the AUIS browser is concerned.

I personally think that we need to play to our strengths, and take advantage of other technologies as appropriate. The ability to create an integrated system, based on our existing technology is very powerful, and we may or may not need to have a tightly coupled browser in order to fully exploit it. A good start would be to create a version of the above 'warp to', and 'link button' procedures that would optionally launch Netscape or Mosaic. From there, it would be interesting to look at other possible interactions. Given the reasonably powerful and well defined Netscape client interface, AUIS may well be the proper tool for integrating it into an larger user environment.

As far as the browser itself, we need to find out exactly who would use it, why they would do so, and which features would be the most important to them. Given this knowledge, we might be able to create a system that incorporates the browser with the rest of ATK into something that is beyond what they would get from Netscape, without having a browser that needs to keep up with it's commercial competitors. If we conclude, however, that the above mention ATK-(Netscape/Mosaic) links really do create the same opportunities, without imposing unreasonable obstacles, then efforts should be put into how best to exploit such a meld, rather than trying to support 'yet another browser'.

---

[1] i.e. "What You See Looks Real Neat", a term coined by David Rosenthal for text that was formatted according to window size and user preferences, as opposed to paper requirements.