SurfIt! - A WWW Browser

Steve Ball
PASTIME Project
Australian National University

SurfIt! - A WWW Browser

Steve Ball

PASTIME Project
Cooperative Research Centre for Advanced Computational Systems
Australian National University
ACTON 0200 Australia
Tel: +61 6 249 5146
Fax: +61 6 249 0010
EMail: Steve.Ball@surfit.anu.edu.au

Abstract

World Wide Web (WWW) browsers have, until recently, only offered static features and media types. A new generation of WWW browsers now becoming available are offering technologies to remove these limitations and allow dynamic extensions to the browser application as well as dynamic media types or active message content.

SurfIt! is an example of these "next generation" WWW browsers. It offers safe execution of downloaded Tcl/Tk applets[1] to support active message content, as well as hypertools to dynamically extend the browser at run-time.

The latest version of SurfIt! is written entirely as Tcl script code, making it highly portable, user-customisable and extensible

Keywords: World Wide Web, WWW, browser, Safe-Tcl, applets.

Introduction

SurfIt! is a new World Wide Web user agent (aka browser) which has been implemented entirely using Tcl/Tk script code. It supports most of the features usually found in Web browsers today - document retrieval using the HTTP protocol, graphical display of HTML documents, inline graphics, hypertext navigation, and so on. SurfIt! also includes many standard browser features such as local caching of documents, incremental document loading and display and concurrent document downloading. As a Tk application, it is easy to construct applications which can communicate with SurfIt!, using the send command, to implement the concept of hypertools. SurfIt! provides direct support for creating tools such as these.

However, the most interesting aspect of SurfIt! is its ability to execute any Tcl/Tk script, which may be downloaded from a remote server, within the context of the browser. These mini-applications, or "applets", are evaluated in a separate, safe interpreter to ensure that they do not conflict with any other Tcl code and that they cannot damage or compromise the user's computing environment in any way.

This paper discusses the implementation of the SurfIt! World Wide Web browser, including the motivation

[1] a "mini-application".

behind its development and a brief synopsis of its early history. Later sections describe the functionality offered by the browser and its internal architecture. Issues concerning the handling of Tcl applets and hypertools will be discussed and finally future goals will be outlined.

Motivation and History

The development of SurfIt! began in early 1995. While developing advanced Web-based hypermedia datasets it was found that active message content was needed to deliver the required user interface functionality. Also, there was a requirement to develop a continuous media[2] playback system which was tightly integrated with a Web browser, and yet loosely coupled [Ball95].

A possible solution to these requirements was to use Tcl/Tk to implement an active message content system as well as taking advantage of Tk's send facility to implement hypertools for handling continuous media. There were several problems facing this approach. The only Tk-based WWW browser then available for displaying textual content was tkWWW v0.12 [2]. This version of tkWWW was based upon Tk 3.6 and so it could not display inline graphic images, which makes it obsolete when compared to modern WWW browsers. Although tkWWW has the capability of executing downloaded Tcl scripts, it lacked the security features of Safe-Tcl to ensure system security. Another alternative would be to use WebRunner (now Hot Java), which used liveOAK (now Java) as its programming language. It was decided not to use this system since at that time the windowing toolkit available with liveOAK lacked many necessary features found in Tk

To address these problems, in February 1995 I created a prototype Web browser, TkWeb. Similarly to tkWWW, TkWeb used the CERN Common Code Library (libwww) to provide network protocol and content type handlers, in particular a HTML parser. This parser was customised so that it would create a list of Tcl procedure calls. The Tk application supplied the procedure definition. When the libwww HTML parser parsed a document, it would pass the generated script to the Tk application which would then evaluate it. This resulted in the application-supplied procedures being invoked to render the document. I found integrating the CERN Common Code Library into a Tk application to be very difficult.

In April 1995 Stephen Uhler [Uhler95] implemented Hippo[3]: a WWW browser mostly written as a Tcl (version 7.4) script. At that time Tcl lacked network connectivity primitives, so Hippo used a HTTP protocol handler written in C. Hippo introduced the now infamous 8 line Tcl HTML parser. This parser filters HTML markup into a Tcl script which is then evaluated by the application to render the document. Again, the parser generates a script with calls to procedures that the application is expected to supply. Stephen released the HTML parser/renderer publically circa May 1995 as html_library.

In late April/early May 1995 I received a copy of Stephen's html_library package as well as Jacob Levy's [Levy95] stcl package - a Safe-Tcl extension to Tcl version 7.4. Given the difficulties I encountered in attempting to make use of the CERN Common Code Library and that a pure-Tcl HTML parsing/rendering subsystem was now available I decided to implement a new Web browser completely in Tcl script code. The only missing functionality at that time was network connectivity, so the popular Extended Tcl package was used to provide low-level network access (another possibility would have been to use Tcl-DP). In addition, the table geometry manager from the BLT distribution was used to implement HTML tables.

Browser Features

The latest version of SurfIt!, version 0.5 [Ball96], is based upon Tcl version 7.5 and Tk 4.1. Since this version of Tcl/Tk includes network socket support and the grid geometry manager there is currently no requirement for any other extensions to be installed.

Below is a brief list of the major features of SurfIt.

- HTML v2.0 (RFC 1866) compliant.

[2] time based media, such as audio or video

[3] short for "Hypocratic"

- partial implementation of HTML v3.2 [Raggett96c].
- partial implementation of TABLE draft specification [Raggett96a].
- GIF, PPM and X bitmap graphics formats supported and may be inlined. Support for JPEG graphics format has been prototyped, but requires a robust photo image type format handler.
- Network protocol handlers for HTTP v1.0 and FTP. file: and mailto: URLs are also supported.
- Asynchronous, concurrent document downloading with incremental document display.
- .mailcap processing for “Helper Applications”.

User Interface

The primary focus of a Web browser is the display of a hyperdocument (which may include embedded documents). Each hyperdocument is displayed in a separate hyperwindow. There will be various functions that may be performed upon hyperdocuments, for example scrolling or activating hyperlinks, and functions for hyperwindows, for example loading new documents, history navigation, controlling applets and so on. The browser’s user interface must provide controls for all of these functions. In addition there are functions that are not related to any hyperwindow, and these are termed browser-level functions. Browser-level functions include creating new hyperwindows, manipulating the local cache, setting user preferences and so on. SurfIt! explicitly separates the user interface for hyperdocuments and hyperwindows from the interface for browser-level functions by placing only browser-level controls in the main window, and each hyperwindow is placed in a toplevel widget. This approach eliminates redundancy in the user interface when the user has more than one hyperwindow open.

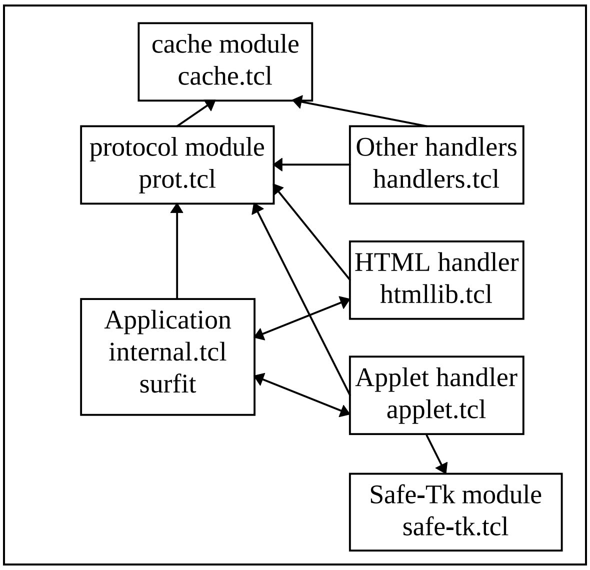
Figure 1: The SurfIt! main window.

Figure 2: A hyperwindow

Since SurfIt! supports the execution of untrusted applets, and it allows those applets access to the Tk Text widget of the hyperdocument into which they were loaded, it is of vital importance to provide a user interface where there is always a means by which the user can control all applets. Safe-Tk, described below, restricts an applet’s use of Tk to a widget sub-heirachy rooted at the hyperdocument’s Text widget or a separate toplevel widget The applet is quite at liberty to compromise any widget to which it has access, and so may disrupt the proper functioning of the browser with respect to that hyperdocument. If the applet is deemed to be behaving inappropriately by the user then by virtue of the design of the browser user interface they may terminate the applet, destroy the hyperwindow and initialise a new hyperwindow as necessary. Hence proper functioning of the browser may be restored.

Architecture

Below is an overview of SurfIt!'s module structure.



The various modules which underly SurfIt! have been designed with reuse as a goal. Other applications are required to access Web documents, for example a web crawler, and these applications are more easily implemented if given access to SurfIt!'s lower-level functionality.

The Protocol Module

The central module is the Protocol module. It manages the loading of documents and their handling by content-type handlers.

When building a World Wide Web user agent, it is a mistake to take the myopic view of Web documents as only being in HTML format, and delivered to the browser using only the HTTP network protocol. In fact, the Web subsumes the FTP and Gopher network protocols for document retrieval, and in addition documents may be retrieved from the local filesystem or by other means - not just HTTP. So the browser developer is faced with the questions: What is a document? How is a document's data delivered to the browser?

A document may be viewed as an atomic data object. Documents may have many media types: plain text, HTML marked-up text, JPEG graphics, MPEG movies, and so on. It is important to note that all of these types of documents are first-class documents and must be able to be loaded as a hyperdocument - they do not have to be embedded in an HTML document. Also, MIME encoding allows a single message to contain several documents (a multi-part message), so the developer should not assume that a message or file is equivalent to a document.

The key issue in loading a document is that the WWW provides a uniform mechanism for specifying a document's location - its Uniform Resource Locator, or URL. An URL has two parts, separated by a colon: the protocol and then the document specifier, the interpretation of which is dependent on the given protocol. For example:

```
          http://surfit.anu.edu.au/SurfIt/
protocol -|       |-host:port        |_path
          mailto:Steve.Ball@surfit.anu.edu.au
```

SurfIt! presents a simple programming interface to a generalised mechanism for dealing with a multiplicity of protocol handlers and document content-type handlers. The Protocol module allows handlers to register themselves and manages the calling of the appropriate handler in response to a request to load a document. The registration system allows new handlers to be added at run-time. The application requests an URL to be loaded using the PRloadDocument procedure. A unique global variable is created to contain relevant information for loading the document. The name of this variable is passed to most of the procedures which subsequently deal with the document. A document's data may be presented to the application in several ways and these are indicated by an enumerated type:

PRdata	the data is a Tcl string.
PRfile	the data is in a file in the local filesystem.
PRfd	the data is to be read from a network channel.
PRpost	the data has been post processed by a content-type handler.

The Protocol module uses fileevent scripts to allow documents to be downloaded concurrently. The fileevent script invokes procedures which manage the process of reading data from a channel, passing that data to the content-type specific processor and rendering procedures, as well as taking of other housekeeping such as cache management.

A protocol handler has two parts (procedures). The handler itself, which is registered with the Protocol module, and a read handler. When an URL specifying that protocol is requested the handler is invoked. The handler commences the document load, and defines the read handler to be used to retrieve the actual data. The read handler is called to read the data of the document as it becomes available, which it returns as its result. A read handler may also prepend data which has been pushed back onto the data stream by a content-type handler. The read handler always writes the data into the cache, which is the only way to handle binary data. In the case of supposedly text documents the newly retrieved data is read back from the cache file to be returned as the procedure result.

Simultaneously downloading documents, and their rendering onto the display, would naturally be done using threads. Unfortunately, Tcl does not provide multithreading. However, concurrent downloading and rendering using fileevent scripts has proven quite effective for incremental document display. Most other modern Web browsers use the same technique, with the Hot Java browser being the only obvious exception.

The Protocol module supplied with SurfIt! includes HTTP and FTP network protocol handlers, as well as a handler for the file: protocol which accesses the local filesystem. A mailto: protocol handler is supplied with SurfIt! in a separate module.

A content-type handler also has two parts: a document processor and a document renderer. The processor procedure is invoked to process the raw data of the document. The result returned by the processor then becomes the input of the handler's renderer procedure. A processor can also register its output with the Cache module as post-processed data. When a document is available in a post-processed form the data is passed directly to the renderer procedure, thus saving some processing overheads. Examples of post-processed data include images, where the data is read into a Tk image, and HTML documents, where the processor procedure filters the HTML data into Tcl commands which the renderer subsequently evaluates, see below.

Content-type handlers for SurfIt! are supplied by the HTML handler and applet handler modules, described below. The other handlers module supplies handlers for plain text documents, graphical image media and "Helper Applications" as defined by the user's .mailcap file.

The HTML Handler

Perhaps the most important content-type handler, and by far the largest component, in SurfIt! is the HTML document parser and renderer. It is based upon Stephen Uhler's html_library 0.3 public release. In fact, to avoid versioning problems when new versions of the library were released the protocol module was designed around the interfaces to the html_library parser and rendering procedures.

html_library had to be modified to work with SurfIt! to support new features included in the browser. The HTML parser could not handle incremently loading a document There were particular problems when a tag was split across two network packets. The parser was changed to detect when this occurred, and to push data back into the input stream to the point where all tags in the data fragment were complete. Also, the parser filtered the HTML data into a single Tcl string which it then evaluated to render the document. A mechanism was provided to set a flag which would cause rendering to stop. However, this mechanism did not work reliably, and there was no means by which the application could cache the parsed document. To solve these problems the parser was modified so that it split the resultant Tcl commands into groups (lists within a list) and returned the parsed document back to the application, which would then evaluate it. At this point the application could cache the parsed data. SurfIt! then evaluated each group of commands in an idle handler to render the document. Performance was improved by populating a Tcl array with each group of commands indexed by a number, thus eliminating excessive list handling in the idle handler.

It is very easy to add handling for new HTML tags to the html_library. For example, Uhler's html_library distribution includes a new <COLOR> tag which is implemented in only 11 lines of Tcl code. Perhaps it is too easy since the temptation to extend HTML in a non-standard fashion becomes almost irresistible! SurfIt! has added handling of tags for proposed new HTML standards - elements for HTML v3.2 [Raggett96c] and tables [Raggett96a]. An example of the display of a table is shown in figure 3.

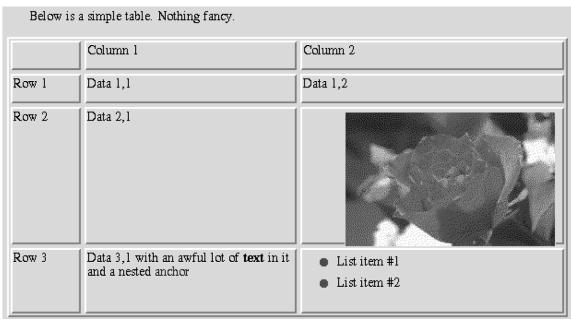


Figure 3 A Table Rendered by SurfIt!

Tables are implemented by creating a new Tk Text widget for each table cell and using the grid geometry manager to layout the widgets in a two-dimensional grid. This made it necessary to add the ability to nest

Text widgets to the html_library. Adding this functionality proved to be a non-trivial task, since some parsing attributes had to be carried through to the nested widget, such as font settings, but some attributes were specific to the nested widget, such as indentation and word wrapping. html_library used a single Tcl array variable to contain state information for the parsing and rendering process. It used the widget pathname of the hyperwindow's Text widget into which the document was being rendered to form this variable name, and the pathname was passed to the rendering procedures. Unfortunately, the pathname was fixed at the time the document parsing started, making it inconvenient to nest widgets.

The new table-enhanced version of html_library splits the state information into two places. Information that pertains to the parsing process and affects all rendering is held in a single array variable whose name is formed from the hyperwindow's Text widget pathname. This variable also contains the stack of nested widgets. so that the currently active widget is easily accessible. Window specific rendering information is held in array variables whose name is formed from the nested widget's pathname.

This approach has been found to be robust when displaying tables and also supports nested tables. This facility will now make it possible to implement a compound document architecture proposed for HTML - the OBJECT element [Raggett96b]. The only problem now with tables is that the Tk Text widget itself needs to be able to automatically resize itself to its contents, and it needs to be able to scroll its contents on a pixel basis, rather than on a line basis, since embedded windows (which are used to display graphic images and tables) appear to the Text widget to be part of a single line and so have undesirable scrolling behaviour.

SurfIt! includes a system for sizing a Text widget: a kludge using the Text widget's scroll commands. To achieve pixel-level scrolling this workaround is used to embed the Text widget in a Canvas widget. The Canvas widget is then actually scrolled by the scrollbars. Scrolling using this method is very slow and cumbersome.

Performance

The Tcl HTML renderer is very slow to display a document due to its highly iterative nature. It is interesting to note that the *parser* is not the cause of slow document display, even though it makes several passes over the document text. Profiling reveals that the critical section of code in the renderer is the HMrender routine which is invoked for every HTML tag in the document. Work is continuing to improve the performance of the rendering engine in the Tcl code. The HMrender procedure has been implemented in C code which makes a substantial performance improvement.

Active Message Content

The applet module provides a content-type handler for application/x-tcl documents. These "documents" are executed rather than visually displayed and allow the implementation of active message content. Such small programs are commonly known as applets. Applets may manipulate a hyperwindow, as in figure 2, or they present their own separate user interface, as in figure 4, or they may do both.

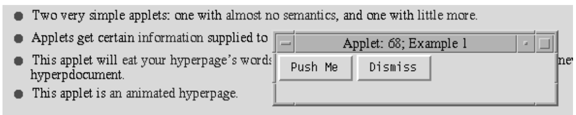


Figure 4: An Applet's Toplevel Window

To execute an applet a safe slave interpreter is created and the applet script is evaluated in that interpreter. A safe slave interpreter has two main advantages: the applet's state and namespace is completely isolated and the applet is prevented from accessing system functions which could result in harm to the user's computing environment. However, an applet which executes in such an environment can provide no useful functions if it has no way of interacting with the user and the browser. Such interaction must occur only in a controlled fashion. Here, a trade-off occurs. The less restriction that is imposed on the applet the more useful functions it can perform, but the risk of damage to the user's computer system increases. SurfIt! seeks to place a level of restriction upon an applet such that the applet can cause no permanent harm.

Security, where the aim of the system is to prevent the state of the computer running the applet from being modified by the applet in an undesirable way, is not the only issue to be dealt with when offering an applet technology. In addition, the applet must be prevented from consuming an unfair amount of system resources, including CPU and memory usage, and network bandwidth. Some resources, such as memory, may need to have some absolute limit set, but for other resources, such as CPU or network utilisation, it is not the total consumption of resources that needs to be limited, but rather the rate at which those resources are consumed. In this case there are no good heuristics available for determining whether an applet has exceeded its fair share. Instead, SurfIt! provides instrumentation which allows the user to monitor resource consumption, and it provides the means to suspend or terminate an applet.

SurfIt! must also concern itself with safeguarding the user's privacy. The overriding policy of SurfIt! is to not make information private to the user available to an applet, since the applet might then transmit that information to a host on the Internet. However, in some circumstances an applet may gather information about a user using information supplied by the browser. An example of this would be where an applet used anchor activation meta-events (see below) to track which Web pages the user has visited. This may be useful marketing information, and the applet is able to transmit it to an Internet host without the user's permission. Preventing this functionality would unduly prohibit other useful, innocuous applets. To reduce the impact of activities such as that described above SurfIt! allows an applet to only gather information about one hyperwindow. When an applet is loaded into a hyperwindow the user should consider that hyperwindow compromised and untrustworthy. It is then the user's choice to start a new, fresh hyperwindow if desired. Also, at any time the user may terminate applets running in a hyperwindow, which will return the hyperwindow to an uncompromised state.

As mentioned above, SurfIt! has several objects for managing browser functions - hyperdocuments, hyperwindows and the browser itself. These objects form a heirarchy in that order, with the browser object being the highest. All applets are attached to a particular object within the browser, the default being the hyperdocument from which they were loaded.. This attachment defines at which level the applet is operating. Applets may change their level, but may only ascend the level heirarchy. Applets are allowed access to certain browser functions and objects depending on their level. So if, for example, an applet wishes to become independent of the hyperwindow to which is was originally attached it must attach itself to the browser, but then it loses access to all hyperwindow objects, including the hyperwindow to which it was previously attached. This ensures the privacy of all hyperwindows.

Certain semantics are defined at attachment levels. For example when a new hyperdocument is loaded all applets attached to the previous hyperdocument are terminated. Similarly, when a hyperwindow is destroyed all applets attached to the hyperwindow (and all child objects) are terminated.

Applets may be associated with HTML fill-out forms (using the SCRIPT attribute of the FORM tag). This forms the fourth and lowest level of attachment. Applets at the form level cannot change level, and cannot attach to other forms within the same hyperdocument. An applet attached to a form has access to all of the form's input items.

Tcl Applet API

To be able to perform interesting and useful functions some applets will need to obtain information from the browser. They will also need to be informed of various "meta-events" that occur during the execution of the browser, such as anchor activation, hyperdocument loads and document display completion, and so on. SurfIt! defines an application programming interface (API) which applets may use to interact with the browser.

The Tcl Applet API is currently under development and will change in the future, but it currently has two methods for applet interaction: an 'applet' command and by means of application "call-ins".

The applet command provides several methods to control aspects of the browser/applet interaction. These methods include commands to query the browser type, return the hyperwindow which loaded the applet and where it was loaded, change the applet level as well as commands to load a new document into the hyperwindow or fetch data given a URL.

Application call-ins are used to inform the applet of a meta-event. A call-in is a procedure prototype that is invoked when that meta-event occurs. The currently defined call-ins include procedures to notify that the applet is to be terminated, that a new form item has been created, that the user has activated an anchor, that the hyperdocument has finished loading, and so on.

Call-ins pollute the applet's procedure namespace and have the potential to cause programming errors. For this reason the call-in mechanism will be replaced in a future version of SurfIt! by a callback mechanism in the applet command.

The Safe-Tk Module

The only means by which an applet can interact with the user is via Tk. However, Tk has not been designed with safety in mind. There are many potential security threats which must be dealt with before an applet should be allowed to create and modify Tk widgets. Also, Tk does not currently handle the multiple interpreters now available in Tcl version 7.5.

Analagous to Safe-Tcl is Safe-Tk [Ball96b]. Safe-Tk is a redesign of the Tk widget set that supports multiple interpreters and that also supports the concept of safe access to Tk. SurfIt! includes a prototype implementation of the Safe-Tk design.

Safe-Tk continues to support only a single widget heirarchy, but different views of that heirarchy can be created by the widget path equivalent of "chroot". Once a safe interpreter has been granted access to a widget subheirarchy all widgets in that subheirarchy are considered to be compromised and the untrusted script may manipulate them as it sees fit. Safe-Tk allows this but ensures that the user is aware of which widgets are compromised (by some osrt of visual indication) and that widgets that have not had access granted to them are not also compromised. SurfIt!'s applet module grants an applet access to the hyperdocument which loaded the applet and also automatically creates a toplevel widget which is aliased to the "." widget in the applet's interpreter.

The Safe-Tk system creates command aliases in a slave interpreter for the widget that it has been granted access to, and for all widgets that are descendents of that widget. Aliases are also created for the various widget class commands. Whenever the slave interpreter configures a widget to perform an action, the Safe-Tk system ensures that the action executes in the slave interpreter. Safe-Tk allows many separate interpreters to each define a widget action and for those actions to all be executed in their respective interpreters, but the current SurfIt! prototype does not yet handle this case.

Safe-Tk also defines command aliases for other Tk commands. Trusted scripts (ie. scripts running in unsafe interpreters) are granted full access to these commands, but restrictions are placed on these commands when they are defined in safe interpreters and some commands are not defined at all.

The most obvious security threat in Tk is the send command. This command is not defined in a safe slave interpreter. Certain other Tk commands cause less obvious security problems. For example, the applet could mount denial-of-service attacks: the grab command would allow an applet to freeze up the entire display with a global grab and with the selection command the applet could continually clear the selection or perhaps set it to some obtrusive message. The various other Tk commands all have particular potential undesirable effects checked and disallowed.

Hypertools

The main focus of SurfIt! is to display World Wide Web documents, ie. it is a WWW user agent (a.k.a. browser). SurfIt! is not a mail user agent, nor is it a USENET news user agent. SurfIt! does not support continuous media (nor is it likely to since Web network protocols are unsuitable for time-critical media). What SurfIt! needs is to be able to call upon some other application to handle those functions when they are encountered, but at the same time to be able to have fine-grained interaction with those tools. This would result in a tight integration of the two applications.

Tk has long held the promise of allowing applications to focus on their main task and communicate with other tools to implement related functions, similar to the way in which small Unix tools can pipe their results to solve complex tasks. This is the concept of "hypertools", made possible by the Tk send command. Hypertools allows applications to remain loosely coupled (ie. they are independent, stand-alone applications), while also allowing for their tight integration. SurfIt! actively supports the hypertool concept.

For all of the tools involved to remain focussed, there will be different hypertools used for different functions. From the perspective of a Web browser there may be hypertools for handling various functions: email, USENET news, and so on, as well as hypertools for display continuous media. For this reason SurfIt! defines classes of functions. However, the good thing about hypertools, like standards, is that there are so many to choose from. For each function class there may be several applications available for use as a hypertool. For example, exmh and TkMail can both handle email.

SurfIt!'s hypertool module provides a registration mechanism to associate an application with a function class. The user may then register her favourite application for a particular class. However, there are further problems. For a given function class SurfIt! needs to know which Tcl commands to send to the hypertool to implement the functions of that class. However, different applications will implement their functions using different procedures and arguments. For example, to compose a new email message (perhaps in response to the user activating a mailto: URL), exmh uses Msg_Compose and TkMail uses mfv:compose. It is essential for SurfIt! to remain independent of these details, so some sort of hypertool protocol must be created for each hypertool class. The hypertool module includes an (incomplete) prototype protocol for continuous media.

Future Developments

There are several improvements planned for SurfIt!. First and foremost is to bring SurfIt! out of alpha release and into beta release. This will mainly involve expanding and stabilising the prototype Tcl applet APIs. Also, general browser functions such as hotlists will be implemented.

The World Wide Web is in a state of rapid and constant evolution of standards. SurfIt! will implement new standards and track their development.

Style Sheets

A major improvement planned for SurfIt! is the implementation of style sheets [Lie]. Style sheets allow the separation of a documents content and structure from presentational information.

The HTML module is well placed to implement style sheets, since it is already well parameterised. However, work will need to be done to handle cascading style sheets and handling nested Text widgets.

Compound Documents

SurfIt! currently relies on the HTML anchor element to embed Tcl applets in a Web page. However, this element is deficient in various ways for this purpose, particularly with respect to nested anchor elements. The proposed INSERT element for a compound document architecture not only improves the method for embedding applets but also provides a generalised scheme for embedding arbitrary media types in a document.

Java

The Java language and associated run-time environment have made a tremendous impact on the World Wide Web. Tcl is in no way incompatible with Java, and the two languages can add value to each other. There are already efforts underway to bring a Tcl/Tk capability to Netscape/Java [Levy]. SurfIt! will complement these efforts by introducing the opposite feature by allowing Java classes to extend a Tcl applet's interpreter. Tcl applet developers will then have a similar facility to that which Tcl application developers have now where C or C++ code may be used to extend the functions of a Tcl interpreter. The Tcl/Java interface [Stanton] will be used as the initial basis for this work.

Conclusion

SurfIt! is a general-purpose World Wide Web user agent that has been implemented entirely as Tcl/Tk script code. It has most of the features of modern Web browsers, including inline images, other HTML v3.2 elements, concurrent document download with incremental display, local caching, and so on. SurfIt! currently lacks certain critical features, such as hostlists and printing, to make it valid choice for casual users, thus restricting its use to research purposes. However, it is planned to add all of the necessary features for general-purpose use in the near future.

SurfIt! supports active message content using applets written in the Tcl/Tk language. These applets are executed in a separate slave interpreter and the host computer is protected from any undesirable actions of the applet. A new API has been created to support the functioning of Tcl applets. An extension to Tk, Safe-Tk, has been designed and prototyped which allows multiple interpreters in a Tk application, including safe interpreters. Although part of the original motivation for developing SurfIt! was the lack of essential functionality in Java's AWT, which is now not very apparent, this work has been continued because of Tcl's benefits in improving programmer productivity, with a view to later integrating the Java/AWT run-time system into SurfIt!.

Hypertools are an essential part of SurfIt!'s multi-protocol, multi-media strategy. Using the hypertool concept, applications such as SurfIt! can focus on their primary task and use standard protocols, based on Tk's send command, to integrate with other hyper-applications.

SurfIt! is being developed and extended in step with developments in the World Wide Web. Most of the major new standards are to be implemented

Acknowledgements

The author wishes to thank Jacob Levy and Stephen Uhler of Sun Microsystems Laboratories for their help in making SurfIt! possible.

I would also like to thank all of the SurfIt! alpha-testers & contributors, in particular Peter Farmer and Tom Tromey.

References

[Ball95] Ball, S. *New Approaches to Custom WWW Interfaces*, Proceedings of the 1995 AUUG/APWWW Conference, Darling Harbour, Sydney Australia, September 1995.

[Ball96] Ball, S. *SurfIt!* World Wide Web browser software.

http://pastime.anu.edu.au/SurfIt/
ftp://pastime.anu.edu.au/pub/SurfIt/surfit.tar.gz

[Ball96b] Ball, S. Safe-Tk Design Document. http://pastime.anu.edu.au/steve/safe-tk-design.html

[2] *tkWWW* World Wide Web browser version 0.12.

http://www.mit.edu:8001/afs/athena.mit.edu/course/other/cdsdev/html/welcome.html

[Raggett96a] Raggett, D. *HTML3 Table Model*. http://www.w3.org/pub/WWW/TR/WD-tables.html

[Raggett95b] Ragget, D, et al. *Inserting Objects into HTML.*

http://www.w3.org/pub/WWW/TR/WD-object.html

[Raggett96c] Raggett, D. *Introducing HTML v3.2*. http://www.w3.org/pub/WWW/Markup/Wilbur

[Lie] Lie, Hakon. Cascading Style Sheets Draft Specification.

http://www.w3.org/pub/WWW/Style/css/

[Stanton] Scott Stanton and Ken Corey. *The TclJava Demonstration*.

ftp://ftp.smli.com/pub/tcl/tcljava-0.1.tar.gz

[Uhler95] Stephen Uhler, Sun Microsystem Laboratories.

Personal Communication.

suhler@eng.sun.com

[Levy95] Jacob Levy, Sun Microsystem Laboratories.

Personal Communication.

Jacob.Levy@eng.sun.com