Cineast - An Extensible Web Browser

Eckhart Köppen
Information Systems and Software Techniques
University of Essen, Germany
Eckhart.Koeppen@uni-essen.de

Gustaf Neumann
Information Systems and Software Techniques
University of Essen, Germany
neumann@wi-inf.uni-essen.de

Stefan Nusser
Management Information Systems
Vienna University of Economics and Business Administration, Austria
nusser@wu-wien.ac.at

Abstract: Cineast is a freely available, extensible Web browser which intends to provide an environment for prototyping new client side Internet technologies. Cineast has built-in support for HTML 3.2, fill-out forms, tables and incremental loading of documents. The browser itself uses the interpreted Wafe environment for implementing the high level control structures. The basic functionality is integrated into the Wafe package and coded in C, which means that the browser gains performance from the speed of compiled code while main aspects of the application can still be changed without recompilation. The network functionality is provided through the integration of the W3C Reference Library. The presentation of HTML documents is handled by the new Kino widget class which provides a flexible and extensible mechanism for parsing and rendering SGML like languages.

1 Introduction

Current development efforts in the domain of W3-applications concentrate on server-side enhancements. The reason for this is the well defined CGI interface, which encourages developers to enhance the server's capabilities. The development of new features on the client side is dominated by a few companies, which have the possibility to integrate new concepts in their browser products. So far, the only way of extending browsers are helper-applications (offering a poor integration with the browser) and plug-ins (being of highly vendor-specific nature). There are many enhancements such as access to new protocols, HTML extensions or peer-to-peer communication, which are impossible to realize this way. As a consequence, the development of such features in a Web browsing environment are mainly in the hand of two or three companies.

As a solution to this problem, we propose our concept of an extensible Web browser called Cineast. It is freely available and can be used as a prototyping environment for new Internet technologies. We achieve this flexibility by several means:

- We use compiled code for providing the basic functionality, but rely on the high-level Wafe [Neumann and Nusser (1993)] environment to implement the main features of the browser. This concept is comparable to that of a 4th generation language. The Wafe environment combines Tcl [Ousterhout (1990)] with MIT's X Toolkit [McCormack et al. (1990)] (Xt) and provides easy means to integrate further packages. Performance critical operations, as for example protocol implementations, HTML parsing or rendering are implemented in C. High level functionality, as user-interface semantics or network request coordination is implemented at the interpreter level and can therefore be modified without recompilation. This feature makes Cineast also an ideal platform for experimenting with e.g. mobile code systems.
- W3C's libwww [Frystyk-Nielsen et al. (1997)] serves as the basis for our network protocol functionality. The integration of libwww into the Wafe package gives us access to the functionality of this library, which is itself of very extensible nature. For example, new protocol suites, MIME-types or transfer encodings can be added in a straightforward manner. There is an ongoing effort of the W3C as well as of other people to keep the library up to date with current developments. The W3C itself is using libwww as a vehicle for

testing protocol extensions (such as PEP [Connolly et al. (1997)]), which made it the logical choice for our project. We have added HTTP over SSL support to Cineast by making use of libwww's protocol extension mechanism. The networking and security features mentioned here are described in more detail in an other paper [Neumann and Nusser 1997].

- For presentation purposes we use a widget class called Kino [Köppen (1996)], which was especially developed with the goal of flexibility and extensibility in mind. The Kino widget class contains a parser for SGML like languages and a rendering engine that is capable of managing arbitrary child widgets (called insets) which can be created in response to unknown tags encountered in the HTML source. Our support of the HTML FORM or IMG tags is completely based on this mechanism. The Cineast browser supports HTML 3.2.

In addition to this, Cineast supports a list of advanced features such as:

- multiple browser instances,
- full support of incremental loading and display (even incremental TABLEs),
- multiple simultaneous requests,
- request folding,
- request-wise transfer monitor,
- scroll linking of HTML source text and rendered display,
- built-in support for GIF, JPEG, PNG, XPM and XBM images.

The two main building blocks (libwww and Kino) will be presented in brief below before the discussion of Cineast itself.

2 The W3C Reference library

Figure 1 shows the main components of the W3C Reference Library and their interactions. A more detailed description can be found in the library's documentation [Frystyk-Nielsen et al. (1997)] or in the paper presented at the Fifth International WWW-conference [Frystyk-Nielsen (1997)]. The *Protocol Manager* is used to coordinate network access for application level protocols. Note that the protocol modules shown in Figure 1 are not part of the library core, although these are the modules, which are shipped with the current version of libwww (version 5.0a). The *Protocol Manager* furthermore provides functions for registering new protocols. We made use of the library's protocol extension mechanism by implementing HTTP over SSL [Netscape Corp. (1996)]. This gives the Cineast browser access to state-of-the-art security technology and allows us to experiment with new Internet security concepts.

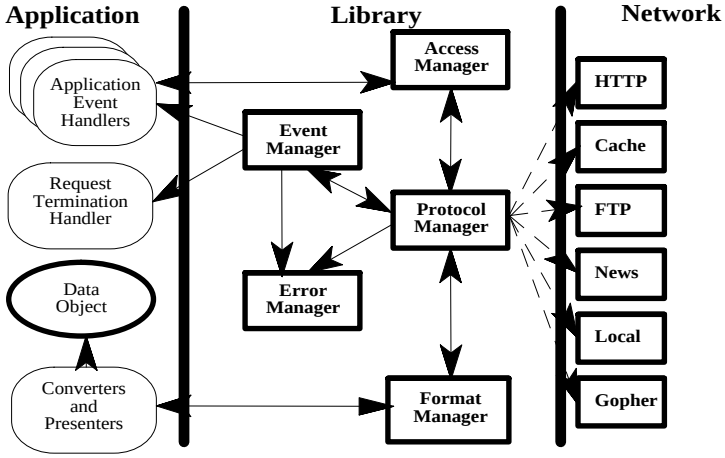

Figure 1: W3C Reference Library Architecture

The *Access Manager* is the main entry-point for applications into libwww's functionality. It comprises several functions for downloading and uploading URLs. Any error messages and warnings which arise during this process are collected by the *Error Manager* and can than be accessed by the application.

The *Format Manager* takes care of any conversions of the incoming or outgoing data. It will handle content-encodings or character-set conversions, as well as the final presentation of a downloaded object to the user, deploying previously registered converters and presenters.

Although the W3C Reference Library comes with its own *Event Manager*, this module is not part of the so called library core. It is the *Event Manager's* responsibility to trigger the protocol modules of libwww whenever data can be read or written from or to the network. This might in some cases dispatch some of the previously registered application-level event handlers, as for example the request termination handler which notifies the application of the request's completion. In our implementation, we actually use the event handling mechanism of Xt which allows us to integrate the handling of network and GUI events. This is of crucial importance for the response behavior of any network application - the same event handling mechanism should be in charge for dispatching user events and network events.

3 The Kino Widget Class

The Kino widget class is an Xt widget class written in C. It implements parsing, formatting and rendering of HTML text. But unlike other tools, it is easily extendible through the Xt callback mechanism. Though the parser of the W3 consortium's libwww and other parsers use this mechanism as well, the Kino widget class goes further by letting the application programmer control most of the internals of the widget. Among these internals are for example the layout information, the HTML source text and much more details. One of the most powerful features is the ability to add insets to the HTML text. These insets can be any kind of widget, even another Kino widget.

The Kino widget has to fulfill three major tasks like any other HTML displaying tool: parse the HTML source text, arrange the parsed elements of the source text and display the elements. Furthermore, proper handling of incremental source text completion is an important feature. Beside these points, the extendibility of the Kino widget requires more functionality:

- provide a clean model for accessing the Kino widget's internals
- interaction with other widgets
- provide a uniform interface for adding custom extensions to the Kino widget

This functionality is realized by three sub-objects: a parser, a layouter and a painter. These objects work on a set of data objects, mainly a list of parsed source text elements (called PData objects) and a list of layouted lines (called Lines structure). Figure 2 shows the interaction of the objects.

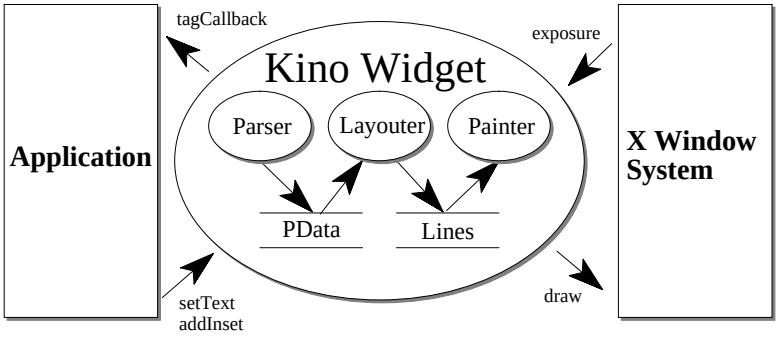

Figure 2: Overview of the Kino Widget Class

The parser is responsible for breaking up the source text into words and tags, which are the only recognized elements. It builds a list of parsed text elements made up of PData objects (these will be discussed further down). Any extension of the core Kino widget class can insert elements into this list during the parsing process such as simple words or more complex style data. The Kino widget itself just adds parsed words to the list.

After the parser (and the Kino extensions) have constructed the PData list, the layouter arranges the elements into displayable lines using the Lines object. The layout of the HTML text is constrained by the available width

and the default text style. The layout process is triggered whenever the available width or the default style changes. Since these conditions occur quite often, the layout process has to be more optimized than the parsing process. The layouter itself optimizes the Lines structure for the painter by calculating as much position data as possible. The painter handles mostly exposure events from the window system but is also used internally for translating screen coordinates to PData elements and source text positions.

The PData objects are the building blocks of the parsed text. The most important elements are words, style and alignment data, table data and insets. These elements can be added when a tag is handled. The parser offers a programmatic interface for the PData list as well as two stacks used for nesting style and alignment data. If the application adds an inset to the PData list, it will be displayed at the current position on the line or aligned to the left or right margins of the text. The Lines structure contains the relation between the PData elements and the corresponding screen positions. It is used by the painter to update the display or to translate screen coordinates to source text positions.

The parsing process is the first point where the extendibility of the Kino widget is implemented: whenever a tag is encountered, the *tag* callback (a resource of the Kino widget with the name `tagCallback`) is invoked. The Kino widget itself does not process the tags further, so the task of handling the tags appropriately is up to the Kino extensions. These extensions can register a callback function for the tag callback using standard Xt functions, which makes the core Kino widget quite simple. The tag's attributes and their values are passed as a parameter to the callback functions.

The standard Kino extensions mostly add text or style data. But by adding insets to the PData list with the `XkAddInset` command, more complex compound documents can be constructed. To demonstrate this feature the Kino widget is extended to handle the `CLOCK` tag from Tcl:

```
proc handleTag {w tag atts} {
  switch -exact $tag {
    CLOCK {
      XkAddInset $w [Clock c $w \
      width 100 height 100 \
      update 1 background pink] bottom
    }
}
```

This tag handler adds a Clock widget whenever the tag `<CLOCK>` appears in the source text. A text like

```
<H1>Clock Example:</H1>
If you are using the Kino widget, you should see a clock
<CLOCK>
```

produces output as seen in Figure 3 where the Clock widget displays the current time and updates itself every second.

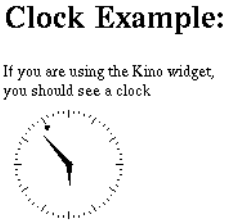

Figure 3: A Clock Widget as an Inset

Another feature of the Kino widget is its ability to change the HTML source text "on the fly", e.g. the Kino widget lets the application programmer change the text after the current parsing position (tag rewriting). By this means it is easy to implement a configurable filter that produces different HTML documents depending on a style guide. Another possible scenario is a client-side interface that allows any script to insert (and change) the source text, e.g. as a result of a database query, or one can handle semantic tags this way.

A semantic mark up like

```
<PERSON>Gustaf Neumann</PERSON>
<AFFILIATION>University of Essen, Germany</AFFILIATION>
```

can be processed can result in an appearance like

Gustaf Neumann is a Person and works for *University of Essen, Germany*

by defining the tag procedure in the browser like

```
proc tag {w tag atts} {
  switch -exact $tag {
    AFFILIATION  { XkChangeCurrentText $w "and works for <I>" 0 }
    /AFFILIATION { XkChangeCurrentText $w "</I>" 0 }
    PERSON       { XkChangeCurrentText $w "<B>" 0 }
    /PERSON      { XkChangeCurrentText $w "</B> is a Person " 0 }
  }
}
```

4 The Cineast Browser

For better code reuse we decided to use OTcl [Wetherall and Lindblad (1995)] rather than Tcl as the base implementation language of the browser. Several classes are used to implement the functionality. A `RequestHandler` handles the life cycle of a request, it has sub-classes for requests for HTML texts and images. Since images are implemented based on the inset capability of the Kino widget class, `Image` inherits from both `RequestHandler` (in order to control the transfer of the file) and `Widget` (to display the image).

The `RequestManager` class keeps track which requests are active per browser instance and aborts requests if necessary. The `HistoryManager` class handles the history of URLs for the handling of the Back and Forward buttons as well per browser instance. Finally the dialog classes are for `mailto:` tags (`MailDialog`), for HTML source browsing and editing (`EditDialog`) and for the transfer monitor (`TransferDialog`) which display transfer statistics on a per request basis and allows termination of single requests. A screenshot of the Browser is shown in Figure 4.

5 Conclusions and Future Work

With Cineast, we present a flexible web browser which is implemented in OTcl and built on top of the Wafe environment. We use libwww for networking functionality and a highly flexible widget class named Kino for HTML rendering. Our basic theme is that we try to combine and to configure efficiently implemented library functions (typically in C) in an as flexible as possible way using Tcl. The flexibility of Tcl (and OTcl) allows to reduce the development time for the sometimes elaborate configurations of the used components and to concentrate on the application tasks. We believe that our environment is one of the most powerful and flexible implementation environments for Web client development currently available. It is straightforward to extend it for:

- electronic commerce (experiment with various electronic payment approaches),
- non-standard Web client extensions (such as mobile code, peer-to-peer document exchange),
- for the development for embedded or specialized browsers (e.g. for certain application domains) and improving Web accessability for user groups such as handicapped persons, or as a
- platform for an Intranet development environment (supporting enabled forms, applets, database access, integration of push-model (email) and the pull model (WWW)), etc.

Our environment incorporates the basic security infrastructure necessary for such projects together with non-standard techniques (such as full tag handling and tag rewriting or insets) which provide more flexibility than plug-ins can offer. Future work includes the use of style sheets like CSS and the implementation of XML to further enhance flexiblity.

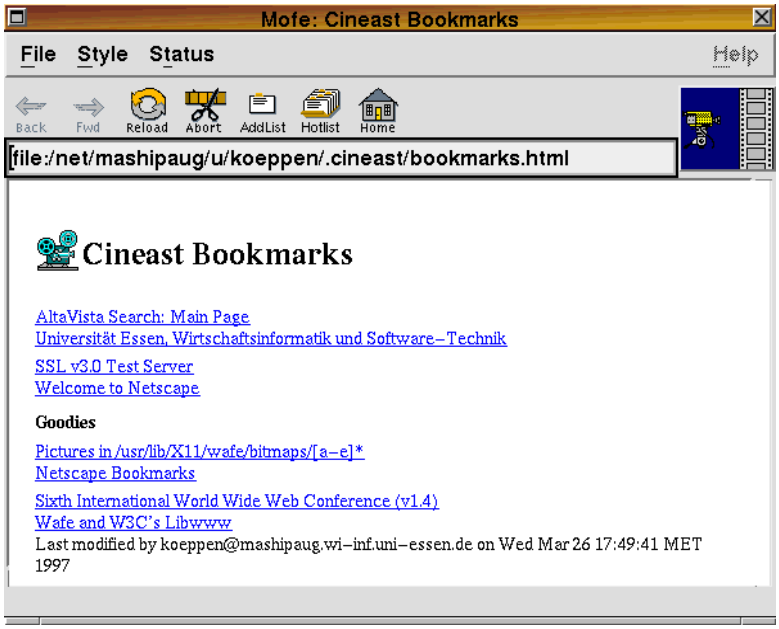

Figure 4: The Cineast Browser

6 References

[Connolly et al. (1997)] Dan Connolly, Rohit Khare, Henrik Frystyk-Nielsen: *PEP: An extension mechanism for HTTP*, work in progress. See http://www.w3.org/TR/WD-http-pep.html in the W3C technical reports section (http://www.w3.org/TR/), July 1997.

[Frystyk-Nielsen et al. 1997] Henrik Frystyk-Nielsen: *Libwww - The W3CReference Library*, http://www.w3.org/Library/, April 1997.

[Frystyk-Nielsen (1997)] Henrik Frystyk-Nielsen: *W3C Reference Library*, position paper for the workshop: "Programming the Web - a search for APIs" at the Fifth International WWW-Conference (http://www.cs.vu.nl/~eliens/WWW5/papers/W3C.html), April 1997.

[Köppen (1996)] Eckhart Köppen: *Entwicklung eines erweiterbaren Widgets zur Anzeige von HTML-Texten*, Master's Thesis, University of Essen, Germany, 1996.

[McCormack et al. (1990)] Joel McCormack, Paul Asente, Ralph Swick: *X Toolkit Intrinsics-C Language Interface*, Massachusetts Institute of Technology and Digital Equipment Corporation, 1990.

[Netscape Corp. (1996)] Netscape Communications Corporation: *The SSL Protocol*, http://home.netscape.com/eng/ssl3/ssl-toc.html, March 1996.

[Neumann and Nusser (1993)] Gustaf Neumann, Stefan Nusser: *Wafe - An X Toolkit Based Frontend for Application Programs in Various Programming Languages*, USENIX Winter 1993 Technical Conference, San Diego, California, January 25-29, 1993.

[Neumann and Nusser (1997)] Gustaf Neumann, Stefan Nusser: *A Framework and Prototyping Environment for a W3 Security Architecture*, Proceedings of CMS'97, Chapman & Hall, September 1997.

[Ousterhout (1990)] John K. Ousterhout: *Tcl: An embeddable Command Language*, Proceeding USENIX Winter Conference, January 1990.

[Wetherall and Lindblad (1995)] David Wetherall and Christopher J. Lindblad: *Extending Tcl for Dynamic Object-Oriented Programming*, Proceedings of the Tcl/Tk Workshop '95, Toronto, July 1995.