Patching onto the Web:
An Emerging CL-HTTP Success Story

Byron Davies
Motorola Logic and Analog Technology Group
R15546@email.sps.mot.com (602) 655-2911 Mesa, Arizona

Victoria Bryan Davies
Industrial Lisp and Magic
vdavies@pobox.com (602) 759-8228 Phoenix, Arizona

Introduction

This paper reports on the successful application of dynamic object technology in rapidly putting a large factory database onto the web. MIT's Common Lisp web server software, CL-HTTP [Mallery 1994], was used to convert a standalone Lisp application into a nearly instant web server. Besides the CL-HTTP software itself, the conversion required a total of about 100 lines of new code, including one new class, two new explicitly exported URLs, and the modification of 5 existing methods and functions. The conversion did not require either recompiling or rebuilding the existing application. In effect, thanks to dynamic object technology, the conversion from single-user application to web server amounted to applying a patch.

The web application is part of an ongoing project to develop automated tools for scheduling wafer fabs (factories in which semiconductor wafers are fabricated) in Motorola's Semiconductor Products Sector. The project is grounded in the belief that manufacturing managers and associates need an explicit, accurate, detailed, and timely model of the factory and its past, present, and future behavior. When completed, the resulting tool set will have six major components: fab model, fab model extraction software, fab browser, historical data browser, scheduling system, and execution system.

The Fab Browser

The Fab Browser is a relatively mature application. Building on [Davies 91], the Fab Browser is a textual outline browser, implemented in the style of Dired within MCL's Fred version of Emacs. In the browser, an object is displayed as lines of text, each representing a component or attribute of the project. Some of the text lines represent other objects, which may in turn be expanded into multiline displays. Though very general, the Fab Browser has been highly customized for displaying wafer fab data, with specialized views of the fab model, scheduling data, and historical data. Using expansion methods that optionally depend on the customer for the display, the Fab Browser is further customized to the needs of each fab, and even to the needs of individuals within the fab,

The basic implementation of the Fab Browser is straightforward. The top level is a function which initializes the Fred buffer with lines of text representing major structural components of the wafer fab and operational data. Each line is inserted by a function called `insert-line`, which inserts a line of text in the buffer and associates an object with the line of text. When such a line is double-clicked, the Fab Browser invokes the `expand` method for the object associated with the line. Each object class in the Orchestra factory representation has an `expand` method, which inserts multiple lines describing the object into the buffer. Hence, information is displayed in the Fab Browser by repeated handoffs between the `expand` methods for objects and the `insert-line` function.

Besides the basic expand capability, the Fab Browser offers other ways of extracting information about a selected object, mechanisms for contracting previous expansions, and a mechanisms for creating a hardcopy report of data that has been expanded. The Fab Browser also offers mechanisms for text filtering of potentially large expansions.

Users find the Fab Browser powerful and easy to use. They like being able to get a high-level summary of fab data and then expand the summary to arbitrary levels of detail using only the mouse. They like the speed of the browser: expansions happen very quickly because tens of megabytes of manufacturing data are cached, in a network of CLOS objects, within the browser application.

Because of the large amount of data cached within it, the browser application runs in 70 to 100+ megabytes of memory, depending on the amount of historical or projected data. This was a significant roadblock to widespread use of the browser, both because of the amount of RAM needed to run such applications efficiently and because of problems of keeping large applications up to date on multiple desktops, some connected by modem lines. The situation demanded a client-server solution, but resources were not available to create either a customized server or a customized client.

Enter the Web

Late in 1995, we began investigating CL-HTTP as a possible means of providing web access to Orchestra data. For a time, because of problems with the MCL implementation of IP/TCP networking, we could not get CL-HTTP to operate properly in our large-address-space MCL applications. Once John Mallery reimplemented MCL's MacTCP interface in March 1996, we were able to make progress on converting Orchestra to the web. The goals for the web implementation were to (1) provide access to the same data, (2) preserve the look and feel, which had already demonstrated its effectiveness, (3) preserve performance, and (4) eliminate dependence on a heavyweight, expensive user platform in favor of lightweight, inexpensive client software.

The result: once the MacTCP problems were resolved, we ported the outline browser to CL-HTTP in about one person-week. The goals for the web implementation were achieved for the most part. Fab data is now accessible through a web browser to anyone in the wafer fab with a computer, whether Mac, PC, or Unix. The look and feel of the web interface are somewhat different from the original single-user version. The main difference is due to the inability to do in-place outline expansion. The single-user application uses multiple levels of nested indentation (like the twist-down behavior of the Mac Finder under System 7). The web version brings up a new window for each expansion (like the Finder under System 6). Per character displayed, the web interface is slower than the single-user interface, but feels like typical web access. Both network delay and CL-HTTP overhead appear to be factors.

Thanks to CL-HTTP, the implementation was almost a trivial undertaking, once we understood what we needed to do. After loading the CL-HTTP software into the application, the port from single-user to server required about 100 lines of new code, with significant changes to only five functions in the original implementation. As described, the original was implemented as recursive hand-offs between the `expand` methods, which know how to format the complete display of an object, and the `insert-line` function, which formats each line of the expansion. The main changes were to put the CL-HTTP server in charge of issuing the expand methods, through an appropriate "search URL," and to force `insert-line` to emit HTML to the appropriate stream when serving the web. The HTML sent by `insert-line` includes an URL which can be used to expand the object associated with the line.

To serve multiple users, the server generates, for each user, an instance of a new class, a connection object. The connection object maintains an index of the URLs to which the user has been given access and to the objects referenced by the URLs. URLs encode both the connection ID and an integer index to Lisp objects that have been made visible to the user by `insert-line`.

Although our web implementation of the Fab Browser is still in prototype form, it promises the client/server solution that we long needed. CL-HTTP and MCL have provided a strong foundation for rapid progress.

First, CL-HTTP provides - in our usual Common Lisp environment - the advantages of an embedded web server: (1) direct access to data, without the need for an intervening CGI-style interface, (2) access to existing Lisp software, including format, (3) easy compatibility between the single-user, Fred-based interface and the web interface, and (4) higher performance than a more loosely coupled solution.

Second, MCL (and Common Lisp in general) provides the advantages of dynamic object technology. Because of this, we were able to create a web server by patching the single-user application. We did not need to recompile or even rebuild the original application. We simply loaded the CL-HTTP software and the few needed changes into the existing application - and we were on the web. Dynamic object technology provided the most tangible benefit in modifying the properties of the `expand` methods to output to the web. To change all of the `expand` methods (for dozens of different classes), we needed only to change the default `:before`, `:after`, and `:around` methods for expand.

Our near-term plans include capturing additional aspects of the Fab Browser, such as filtering of output through text-matching and the transmission of graphical output (e.g., column and bar charts) across the web. The next step will be to recapture client-mediated, in-place, indented expansion, probably with Java.

Conclusion

Dynamic object technology wins again, with CL-HTTP as the vehicle. CL-HTTP, layered on top of an existing object-oriented Lisp application, has made it possible - even easy - to rapidly retarget a valuable data browsing application to the web.

Acknowledgments

The authors are grateful to John Mallery for CL-HTTP, Digitool for MCL, Steve Mitchell for ongoing discussions, and Bill Chapman at Motorola for providing an environment for innovation.

References

[Davies 1991] Byron Davies. Process Specifications and Process Knowledge in Semiconductor Manufacturing. Stanford University EE Ph.D. Thesis, 1991.

[Mallery 1994] John C. Mallery. "A Common LISP Hypermedia Server". In Proceedings of The First International Conference on the World-Wide Web, Geneva: CERN, May 25, 1994.