On The Internet Nobody Knows You're a Lambda

Opportunities and Challenges for Developers of Lisp-Based Web Applications

Brent Benson*
Oracle Corporation

Abstract

Today, most large-scale programs are web-based and communicate with users through an HTTP server. In this new server-based world, issues like application delivery, self-contained executable creation, support for native windowing APIs, runtime library size and other issues that have prevented the widespread adoption of more diverse and higher level programming languages are becoming less and less important. This paper attempts to point out some of the opportunities and some of the challenges that face developers who want to use languages like Lisp and Scheme to write the next generation of web-based applications.

Introduction

Just about any commercial software developer who is drawn to non-mainstream programming languages has had to deal with resistance to choosing a language for a project that isn't considered a safe choice, suffering from the "Nobody ever got fired for choosing language X" effect. Here are some of the reasons we have heard for not using a Lisp-like language:

- "Lisp is not efficient enough."
- "We won't be able to find programmers that know Lisp."
- "The runtime system for Lisp is too big and you can't generate standalone executables."
- "Lisp uses a garbage collector, and garbage collectors are less efficient than manual storage management."

While many of these reasons didn't hold much water ten years ago, they are even less tenable today. When flat-out algorithmic performance is critical Lisp implementations can produce code at least as fast as Java and can often compete with C and C++; there are many strong Lisp programmers around and the learning curve for high quality engineers is pretty small; delivery issues like runtime library size and small executables are irrelevant on a shared server; and pause times for modern generational collectors are often less than network latency times, not to mention the fact that Java, a garbage collected language, is one of the most popular on application servers.

This paper attempts to take a balanced approach and describe the opportunities and advantages to using Lisp-like languages to implement web-based applications and also to describe challenges and pitfalls that might await implementers.

*The opinions expressed in this paper are those of the author and do not reflect those of Oracle Corporation.

Opportunities and Advantages

You're In Control

Unlike traditional desktop applications, the code for a web-based application resides on a server under the control of the application provider. Because customers are using a simple web browser to access the application, they don't care which technology is used to implement the application as long as it does what they need it to do in a performant and user-friendly manner. This simple fact has vastly increased the opportunities to use a more diverse set of operating systems, programming languages and libraries. If you are starting a company to provide a web-based application it makes sense to consider the full range of operating systems and high-level language implementations, including open source implementations as they can provide a significant cost savings for small startups[1]. Technical decision makers can choose productivity and reliability over market share and consumer acceptance. The separation of concerns between the customer and the implementer provided by the HTTP server allows the implementer unprecedented technological flexibility. You should take advantage of this flexibility—your competitors will.

Interactivity: Better Prototyping and Debugging

One of the key advantages of using a Lisp-like language for developing a large application is the interactive nature of most Lisp development environments. The read-eval-print loop allows for quick prototyping and testing and integrated debugging allows for quick pinpointing of a problem, experimenting with fixes, making tentative changes, and immediately testing the new code. Lisp systems usually allow for code changes without relinking so that a server can be updated with a fix and a customer can try the fix in minutes, rather than days[2].

Better Languages Attract Better Programmers

It is indisputable that there are more Java programmers than Lisp programmers. But when you start a company, you don't need thousands of programmers—you need a few good programmers. Programmers who end up learning Lisp don't usually do so in order to earn more money or get a better job, they do it because they like programming and are driven to find and use better tools. These are exactly the kinds of programmers that companies should want to hire[4]. Application development and maintenance is expensive and better tools, better programmers, and better code will prevail in the end.

Closures and Continuations

A most promising area that has been explored both theoretically[7, 8] and practically[3, 5] is the idea of using closures and/or continuations to manage state in web-based applications. Those of us that have developed large scale web applications can attest to the fact that handling arbitrary user navigations in stateful web application flows can be a frustratingly hard problem. Users can use the browser back, forward and refresh buttons or use the browser history mechanism to jump to an arbitrary URL visited during a session.

The basic idea behind the continuation capturing technique is to use first class continuations or specially created closures to represent the remaining computation within an application flow. This continuation state is associated with particular page view within the user's server interaction. If the

user chooses to jump around to previous pages using the various browser navigation mechanisms, the current state and remaining computation is readily available in the continuation. The big advantage to this approach is ease of programming and predictable and safe semantics for the user.

Macros: Natural Domain-Specific Language Extensions

There is a great deal of redundancy in any large web application. Pages share structure; editing flows share common steps; errors need to be presented to users in a consistent way. A popular technique for dealing with certain kinds of application redundancy is to allow declarative creation and sharing of structure using domain-specific languages like stylesheets, templates, Java Server Pages, etc. A major problem with many of these declarative languages is that they require a programmer to completely switch gears into a different syntax and into a different set of development tools. In addition, the abstraction level for these languages is usually fixed and resists attempts to add or subtract layers as appropriate for a specific application.

The advantage for Lisp-like languages is that they are designed to allow easy creation of language extensions, primarily through domain-specific macros. By allowing the creation of language extensions within the language itself, programmers are encouraged to develop appropriate levels of abstraction while constructing the application, rather than requiring a completely different set of skills to create a template language or tag library. Programmers can use the same development environment and tools for creating and debugging their extensions, and they can add or subtract abstraction layers as appropriate. Many Lisp-based web servers provide a built-in set of macros for declaratively describing web content which can be added to and extended with application-specific constructs, allowing much simpler and less error prone application construction and enhancement.

Challenges

Perception

If you are starting a company yourself and don't need outside funding, you probably don't need to worry about what anyone else thinks of your technology stack. For everyone else, there will be a need to communicate your technical vision to company leaders or investors who might share some of the concerns referred to elsewhere in the paper (efficiency, support, etc.). You'll have to make the decision about whether it makes more sense to describe the details of your technology stack and convince the stakeholders that the advantages outweigh the disadvantages, or whether you want to stress the standards-based interfaces to the system (HTTP, XML, SQL, etc.) and gloss over the implementation details.

Integration with Partners

It is not always the case that your customers reside on the other side of an HTTP server. In particular, companies often need to integrate with existing applications, or provide an application that can be integrated into a partner's existing setup. In situations like these, there can be many constraints on what tools and technologies your partners will accept. One solution is to define your integration in terms of web services through an HTTP layer; in that case you can probably still keep the flexibility you need to use your preferred set of tools. Otherwise, there are some middle-ground solutions like

using a Lisp system connected through Apache extensions (such as `mod_lisp`) or a Java-based Scheme system integrated with a Java-based application server.

Support

There are arguments about whether you can get better support by using open source tools or buying commercial software, but if you prefer the commercial software approach there are a limited set of choices. While there are several commercial Common Lisp implementations, and one major commercial Scheme implementation, the HTTP and application server components available are not commercial products. There are some individuals and companies that will provide commercial support for open source Lisp implementations like CMU Common Lisp and Steel Bank Common Lisp and that may also provide some support help for Lisp-based web servers.

Not Invented Here

There can be a tendency for companies and developers that are using non-mainstream tools to reject new tools, libraries, or systems because they were "not invented here." Of course there is often less of a need to use outside tools and libraries when your language provides a high level of functionality, but it is important to accept outside ideas and technology when they make sense. It helps to have a person or persons who are responsible for keeping up with industry trends and look for ways that developing technologies can improve the product, or the tools used to construct the product. A balance needs to be struck between needless change and constant reinventing of the wheel.

Conclusion

Customers that use web-based applications don't care what sort of implementation technology is used to create an application as long as the application fulfills its function. This separation of concerns allows application implementers unprecedented flexibility in terms of the tools used to implement and deliver applications. There are many advantages to using a Lisp-based language as an implementation tool for a web-based application including opportunities for better programmer productivity and more robust functionality.

There are several examples of highly successful web-based applications implemented in Common Lisp including Yahoo Store[2] and Orbitz[6]. There are probably other Lisp-based web applications that are currently flying under the radar (Yahoo Store and Orbitz were not recognized as Lisp-based applications for some time—the implementers didn't feel the need to make a big deal about the technology; they just wanted the advanced capabilities).

There are quite a few tools available to support web application building in Lisp and Scheme. There are several web servers written directly in Common Lisp including `CL-HTTP`, *AllegroServe*, and *Araneida*. There is also the `mod_lisp` plugin for Apache that allows an Apache web server to connect with a running Lisp process. The MzScheme system provides a native web server and libraries for implementing application server functionality, and many Java-based Scheme systems including *SISC*, *JScheme* and *Kawa* allow serving up web pages inside any web server that supports Java Servlets.

While Lisp is almost surely not the correct implementation language for every web-based application, it deserves a fresh look by those looking to create the next generation of innovative web applications.

References

[1] Eric Auchard. *Plugged in - Next Big Tech Ideas May Be Small Ones*. `http://www.reuters.com/audi/newsArticle.jhtml?type=technologyNews&storyID=8067402`. April 2005.

[2] Paul Graham. *Lisp in Web-Based Applications*. `http://lib.store.yahoo.com/lib/paulgraham/bbnexcerpts.txt`. April 2001.

[3] Paul Graham. *Beating the Averages*. `http://www.paulgraham.com/avg.html`. April 2003.

[4] Paul Graham. *The Python Paradox*. `http://www.paulgraham.com/pypar.html`. August 2004.

[5] Paul Graunke, Shriram Krishnamurti, Steve Van Der Hoeven, Matthias Felleisen. *Programming the Web with High-Level Programming Languages*. In *European Symposium on Programming*, pages 122-136, 2001.

[6] Carl de Marcken. *Inside Orbitz*. `http://www.paulgraham.com/carl.html`. January 2001.

[7] Christian Queinnec. *The influence of browsers on evaluators or, continuations to program Web servers*. In *ACM SIGPLAN International Conference on Functional Programming*, pages 23-33, 2000.

[8] Christian Queinnec. *Inverting back the inversion of control or, Continuations versus page-centric programming*. Technical Report 7, LIP6. May 2001.

About the Author

Brent Benson is the creator of several open-source languages implementations including `libscheme` and *Marlais*. He has worked as a compiler implementer at Harris Computer Systems and as a designer and implementer of the domain-specific configuration language *GSL* at ICAD and Concentra Corporation. He is currently a Group Manager at Oracle Corporation working in the area of configuration technology.