Firefox's developers are working to improve the performance of its JavaScript implementation. Although this obviously helps users visiting JavaScript-heavy websites, it also allows the Firefox developers to migrate more and more of the browser itself from C++ to JavaScript, a much more comfortable and flexible language for the problem. In this sense, Firefox's architecture is evolving to look more like that of Emacs, with its all-Lisp Controller.

This suggests the third and final question, one we can ask of any extension language we encounter: *is the extension language the preferred way to implement most new features for the application?* If not, what restrictions discourage its use? Is the language itself weak? Is its interface to the Model cumbersome? Whatever the case, these same weaknesses probably affect extension developers in similar ways, leaving extensions as second-class citizens. (One could ask the analogous question of plug-in interfaces as well.) Like Emacs, Firefox places its extension language at the heart of its architecture, a strong argument that the language's relationship with the application has been designed properly.

As an avid Emacs user, but one concerned about its future, I'm especially interested in Firefox because it seems so close to Emacs in many ways: a View that provides automatic display management, a Controller based on an interpreted, dynamic language, and a Model that does everything Emacs's does, and much more. If one were willing to leave behind the accumulated corpus of Emacs Lisp code, a few days' worth of chrome programming could produce a text editor with an architecture very similar to Emacs's, but with a much richer model and a much stronger connection to the current frontiers of technology. The most valuable lessons Emacs's architecture has to teach need not be forgotten.