Crossfire - Multiprocess, Cross-Browser, Open Web Debugging Protocol

Subtitle Text, if any

Michael G. Collins
IBM Research - Almaden
mgcollin@us.ibm.com

John J. Barton
IBM Research - Almaden
johnjbarton@johnjbarton.com

Abstract

We present Crossfire, a protocol designed to enable remote debugging with the Firebug Web debugging tool, and an implementation of this tool as a Firefox extension. We also present an architecture in which the user interface of Firebug is separated from the backend debugger into separate processes connected via the Crossfire protocol.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

General Terms term1, term2

Keywords keyword1, keyword2

1. Introduction

Web Applications continue to grow in size and complexity. Asynchronous background data download (AJAX) started the surge. This led to the emergence of common toolkits and libraries for Javascript which drove performance increases in Web Browsers fueling more growth in client-side Web application development. These improvements, combined with new features available in Web browsers shifted investment from server- to client-side. Recent empirical analysis of representative major Web sites shows program sizes in the range of hundreds of kilobytes of sophisticated code.[VitekDynamicJS2010]

To develop and maintain these large applications, programmers and designers rely on numerous tools, most notably Web page debuggers. This paper describes a major re-architecting of the most widely used Web page debugger, Firebug, as a client/server system and in particular the *Crossfire* protocol designed to support its client-server communications. Our description focuses on practical, state-of-the-art issues in an on-going and fast-moving project. Thus we cover details of protocol important for implementation and issues of matching resources to goals important for project management: we must deal with both extremes to succeed.

2. Background

To understand the importance and challenges of the *Crossfire* work we start by introducing Firebug. Released by Joe Hewitt in 2006, Firebug was the first integrated Web debugger. Firebug is a runtime debugger: it directly accesses, responds to, and operates on the running Web browser. Rather than separate views of JavaScript, CSS, and HTML, Firebug integrated its views such that interaction with, for example, an HTML element would cause synchronized views of the CSS rules. Rather than static views of browser state, Firebug included dynamics like network traffic analysis and console logging; rather than read-only views, Firebug allowed live edits where possible so developers could try out changes. The resulting tool became very popular with developers and contributing significantly to the growth in Web applications.

The primary implementation of Firebug is a Firefox extensions, a supplemental software component that loads into the Firefox Web browser. A secondard implementation with fewer features and, in particular, limited support for JavaScript debugging, called Firebug Lite works in multiple browsers. The success of Firebug lead to implemenations of Web Page debuggers in other browsers, including DragonFly for Opera, Web Inspector for Google Chrome and Apple Safari, and the developer tool in Microsoft Internet Explorer. Since 2007 Firebug has been developed as an open source project, with seven major releases.

To give a flavor of the kinds of operations Firebug supports, we outline an example more completely described in Ref.[3]. Suppose a developer wants to understand why a block of text in the Web page turned green while the page was loading. They might use the Firebug "inspect" feature, causing Firebug to display the HTML element under the mouse. They see that the element has a *style* attribute setting the color green. While hovering over the green block of text, the developer clicks down to lock the user interface on the element, then moves to the HTML panel in Firebug and right clicks on the selected HTML element representation. A menu pops up allowing the developer to select "Break On Attribute Change". When the developer reloads the page, Firebug halts in the JavaScript code panel, on the line where the attribute is changed from red to green.

This example illustrates that we will need to synchronize mouse events on the Web page with the debugger UI, identify HTML element representations rendered in the debugger UI with the elements in the Web page and synchronize DOM mutation event handling wit JavaScript execution.

(Refactoring timeline) ======= Web Applications continue to grow in size and complexity. The emergence of common toolkits and libraries for Javascript along with performance increases in Web Browsers fuels growth in client-side Web application development. As more functionality shifts from server- to client-side, the size of Javascript codebases naturally increases. Unfortunately for

developers, the tools to develop, manage, and debug larger codebases have not followed the applications into the Web application development space.

Sophisticated development tools become crucial for understanding and working with larger codebases. Compilers, debuggers, and other tools are often combined to create an Integrated Development Environment (IDE) where most, if not all, development tasks can be performed. Web development tools, on the other hand, often reside in the Web Browser. In 2011, most Web browsers ship with some kind of Web development tools included[**?**], and still more are available as plugins or add-ons. These tools operate directly on the runtime Web application that is loaded into the browser, and do not retain any knowledge of or connection to the original source code. Therefore a developer must manually apply any changes made in one of these tools to his or her source code.

This class of built-in Web development tools allow for a powerful set of features. However, several weaknesses.

Tools such as "Inspectors" and Firebug's HTML Breakpoints[3] enable developers to quickly locate the section of code they are interested in.

The rise of Mobile and Tablet computers which ship with fully functional Web browsers means that Web application developers have additional form-factors to consider.

Tools such as "Inspectors" and Firebug's HTML Breakpoints[3] enable developers to quickly locate the section of code they are interested in.

Crossfire is designed to support the features available in current built-in development tools such as Firebug, while enabling the advantages of a remote debug connection.

3. Design Motivation

As a practical project supporting 3 million users, Firebug drove many of the design considerations behind *Crossfire*. These considerations are a mixture of purely technical issues and issues of open source project management. The main design goals, multiprocess support, remote and mobile debug, and open Web, cross-browser debugging lead to many of the technical design choices. On the project management side we must work with development resources motivated by goals: no matter how much value Firebug users may recieve from a goal, the selection must be limited by the motivation of open source contributors.

Necessity motivated first *Crossfire* design goal, multiprocess support. Soon after the Google Chrome browser was released, the Firefox team at Mozilla began plans to convert Firefox to a multiprocessor design. The Google browser used one controlling process for the application and one process for each Web page. This allows the browser to use the operating system isolation to prevent problems on one page from bringing down the entire application and it allows each page to use a different physical processor on modern multi-core computers [**?**]. Depending upon the Firefox browser plaform changes, a shift to multiprocess could render Firebug unusable. As a practical matter we could not wait to for the new platform to be available: with more than 50kloc of code, only a few full-time developers, and a commitment to continous compatibility with Firefox we had to begin work immediately to insure that our small resource could complete the transition in time to remain a viable project. Therefore we assumed that Firefox would adopt an architecture similar to Google Chrome: a client/server split debugger with a backend in one Web page process and a front end in another process. We believe that this assumption is planning for the worst case: converting Firebug to client/server is a multi-person-year effort but very likely to work with what ever the Firefox team decides to do.

While necessity forced our action, opportunity followed. The client/server choice, if successful, adds two new dimensions to Firebug for users: remote debug and mobile device debug. We expect the value of these dimensions to grow as more developers work in distributed teams and as mobile plays an increasingly important role in Web application development. In fact this value was recognized by the DragonFly Web debugger for Opera well before even the Google Chrome browser. The additional cost of designing for remote and mobile debug on top of a client/server design, primarily mechanisms for specifying the connection addresses, comes with high potential benefit. Moreover, the benefit aligns with directions important to the projects primary open source contributors.

The final goal, open Web, cross-browser debugging, offers even more benefits to Firebug users. Web application developers by definition target all Web users, but the all Web users are not running identical Web platforms. Almost all potential users of a Web site will be running one of few similar but slightly different browsers. The commonality allows Web developers to do most of their work on one browser, then test for differences on other browsers. Of course when this fails, they need to debug the problem on a browser with unfamilar debugging tools. A common debugging tool across the major browsers would help with this common and significant problem.

The benefit of cross-browser debugging comes at a steep cost for the project. Instead of one server and one client, we face at minimum one server for every browser. And for each server we have to deal with both the slight differences in browser implementation of standard Web APIs and potential large differences in how debuggers can connect to the browser. In addition this goal implies that the client and the communication protocol should be built from open web standards to maximize the reuse across servers.

Perhaps unique to open source projects, Firebug might balance the cost of implementing cross-browser debugging support by attracting more contributors interested in this particular goal. That is, by adding this costly goal we can attract new contributors, allowing us to create more total value. In particular new contributors from the Orion project[**?**], joined to create *Crossfire* server for Microsoft Internet Explorer and from the Eclipse project[Eclipse JSDT] to create new *Crossfire* client in Java for connecting to Eclipse.

These design goals created constraints for *Crossfire*. Above we outlined how the multiprocess support lead to a client-server design choice. Support for remote and mobile debug forces isolation of user interface to the client (excepting some small interface for connection specification). The cross-browser goal creates constraints indirectly: to minimize the extra cost of supporting multiple servers we chose to adopt the Google Chrome communications channel (sockets) and wire protocol format (JSON). Neither Firefox nor Internet Explorer had existing servers, so they did not alter our choices. Opera had a server but no one on the open source team planned to work with Opera and the server itself was not open source making implementation more difficult. Since Firebug is already written in JavaScript, JSON format is especially easy to work with and has good performance[**?**]. For the communications protocol, HTTP would be a better choice for the project: the JavaScript support for HTTP is much better than sockets and HTTP works better in practical remote scenarios through firewalls. However we made the judgement that better socket support was coming in future[**?**], support was adequate now, and lowering cost on the Google Chrome server was important.

4. Background

Recently there has been renewed interest in Web browser development and a corresponding increase in development of in-browser Web development tools.

Code that the user wishes to debug may not always be running. TODO

The user may have several user interfaces in which to interact and debug the runtime. TODO

5. Protocol Design

5.1 Overview

The CrossFire protocol is an asynchronous, bi-directional protocol designed to enable the full functionality of the Firebug debugger in a multi-process or remote scenario. Where it was possible, the design of the protocol took cues from existing debug protocols () , as well as common Web technologies (e.g. HTTP[1], JSON[**?**]). However certain features unique to Firebug and to debugging code running inside a Web Browser must be taken into account in the design of the protocol.

Implementations of the protocol differ based on whether the implementation is intended to operate as a client or server. A CrossFire server resides in or is connected to the process which is acting as the runtime platform for the Web page, application, or other code which is to be debugged. This is typically a Web Browser, although supporting other runtime environments is envisioned. A CrossFire client connects to a server in order to receive events and issue requests, typically in order to provide a user-interface for debugging, (e.g. GUI or command-line debugger). It is not necessary for the client and server to reside in the same process or even the same host machine.

5.2 Connection and Handshake

CrossFire does not specify a standard or well-known port. Port agreement is left up to the user, or the client software must start the server listening on the same port it will attempt to connect to.

The CrossFire server listens for a TCP connection on the specified port (greater than 1024). A client wishing to connect sends the string "CrossfireHandshake" followed by a CRLF. The server replies with the same string, at which point the connection is established and the client may begin sending requests and receiving events from the server.

5.3 Message Packets

A well-formed CrossFire packet contains one or more headers consisting of the header name, followed by a colon (":"), the header value, and terminated by a CRLF. A "Content-Length" header containing the number of characters in the message body is required.

The message body is separated from the headers by a blank line represented by a carriage-return character followed by a line-feed character (CRLF). The blank line should be followed by a well-formed JSON string, and terminated by another CRLF. The message must contain a "type" field with the value one of "request", "response", or "event", and a "seq" field which contains the sequence number of the packet.

Example: TODO example packet

5.4 Client/Server Behavior

Once a connection has been established and a successful handshake is completed, the server may begin sending events to the connected client using the same TCP connection used for the handshake. A client may also begin sending requests to the server using the same connection.

5.5 Extensibility

5.6 Contexts

A context in Firebug represents a single Web page.

5.7 Breakpoints

Breakpoint debugging is a standard tool for debugging software at runtime in many languages and environments. The Web Browser environment creates several challenges for designing a remote protocol which supports breakpoint debugging. Firebug also introduces several types of breakpoints which are not present in other environments [3]

6. Implementation

6.1 Crossfire Firefox Extension

The Crossfire extension implements the protocol as an extension to Firefox and Firebug.

6.2 Crossfire Tools API

The Crossfire extension also implements an API, called the "Crossfire Tools API" which enables extensibility of the Crossfire system and protocol.

6.3 Modules

6.4 Browser Tools Interface

7. Discussion

8. Related Work

8.1 GDB

8.2 JNDI/JDWP

8.3 DBGP

8.4 Opera Scope

Opera Scope Protocol [2]

8.5 V8 / Chrome Dev Tools

8.6 wienre

9. Future Work

9.1 Web Sockets

9.2 Multi-user Debugging

A. Appendix Title

This is the text of the appendix, if you need one. I don't

Acknowledgments

Acknowledgments, if needed.

References

[1] Rfc2616.

[2] Opera scope protocol.

[3] J. J. Barton and J. Odvarko. Web page breakpoints. In *WWW2010*, 2010.

References

[Eclipse JSDT] http://wiki.eclipse.org/JSDT/Debug

[VitekDynamicJS2010] Richards, Lebresne, Burg, Vitek, An Analysis of the Dynamic Behavior of JavaScript Programs. PLDI