Chatting on the Web

Martijn van Welie and Anton Eliëns
Vrije Universiteit, Department of Mathematics and Computer Science
De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
email: {martijn,eliens}@cs.vu.nl, fax: +31.20.4447653

Abstract:

This paper will discuss how applications can be integrated in Web browsers. This approach enables cooperative work by using the Web as an entry point for starting applications that can offer more specific functionality for cooperative work. We propose the use of an other communication protocol on top of HTTP for the integration of applications. An example of a chatting page is discussed as a first step towards cooperation.

1. Introduction

The Web is slowly becoming more and more alive. The days of the static pages are numbered and with the introduction of inline applets, the web is becoming increasingly dynamic. In [van Doorn and Eliëns 1995,Eliëns, van Ossenbruggen and Schönhage 1996] we described a web extension for Hush[Eliëns 1995] that, among other things, enables the execution of Tcl code inside an HTML page. In combination with other Hush extensions e.g. a video and an audio widget (the Hymne[van Ossenbruggen and Eliëns 1995] extension) even more interesting applets can be written. HTML documents can be made as dynamic as the used scripting language permits. By now it has become clear that the execution of scripts such as Java[Gosling and McGilton 1995], Tcl[Ousterhout 1993] or Python are a powerful means for building more dynamic features in Web pages. However, cooperative work using the web has its own problems, in particular when only the functionality of standard browsers and servers is used. Applets can offer the functionality that is needed to fill the gap. This paper will discuss how applets can enable cooperative work by integrating arbitrary Hush applications within a web browser.

2. Example -- a chat page

In this section, our example of a chat page is discussed on a conceptual level. The next section will discuss the implementation of the example in our Hush browser.

2.1. Chatting applets

With our work we try to stretch the boundaries of script execution. We created a demo web page that contains a chat applet that chats with other chat applets. By requesting this page a user can join a chat session such as the well known IRC chat sessions. The applet directly communicates with other applets, and does not use CGI scripts or the HTTP protocol (see figure 1). For writing a chat application, the HTTP protocol is not very suitable so we developed a new protocol to support direct

communication. Our browser currently supports Tcl applets so the applet that is contained in the page is written in Tcl instead of Java.

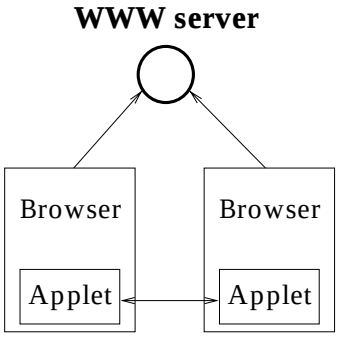

Figure 1: Chatting applets

2.2. Communicating with applications

The applet is a script that uses a chat component. However, since such a chat component is not a standard component in every browser, we modified our browser to setup communication with other applications so it can delegate commands for unknown components. An alternative would be to compile such components into the browser but that would lead to a large monolithic browser what we consider as rather inflexible. Instead, our browser communicates with remote applications to delegate functionality. In this paper we will use the term remote applications for applications that are running on some host (possibly other than the local host) with which a browser can communicate.

In our Hush browser, a connection with a remote application is set-up after the page is loaded, and the browser only serves as a viewer for the interface of the remote application.

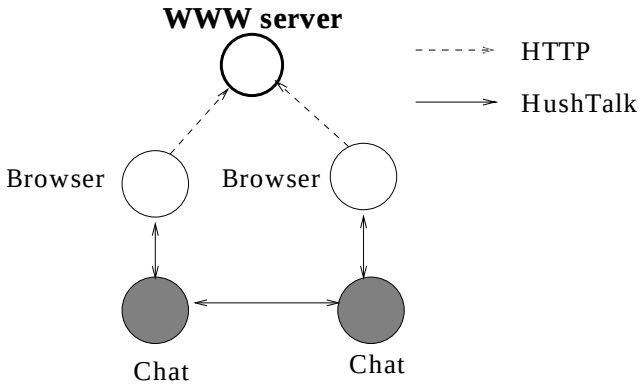

Figure 2: After requesting the chat page

The WWW server has no active role except for initiation, see fig 2. If two users have requested the chat page, two chat applications will be running and the interface will be visible embedded in an HTML page. This example illustrates the use of new functionality that the browser nor the WWW server itself support. The WWW is used only as a general access-point for using applications on the network and to have the web browser act as a viewer for the application interface.

2.3. HushTalk -- communication on top of HTTP

For communication between a browser and a remote application , an additional communication protocol is needed. Such a protocol should at least have the following properties ; it should be fast, have two way communication with flexible message passing and some form of session management.

The first two properties are needed for efficiency reasons and to keep up the performance of the system. The last property is important for cooperative work so applications that participate in cooperative work can reach and be aware of each other throughout a session.The stateless HTTP protocol is not suited for this kind of continuous communication, nor does it offer any session management. Therefore we developed a new communication protocol called HushTalk, currently based on Sun's ToolTalk[Julienne and Holtz 1994].

ToolTalk is a message based system for inter-application communication which is available for UNIX platforms as part of the standard Solaris 2.x distribution. The actual communication in ToolTalk is done by RPCs which make it faster than the HTTP protocol since connections are not opened and closed all the time. It also provides the notion of a session that manages connected applications and message delivery. Choosing an other protocol than HTTP also has the advantage that the HTTP server load is not increased.

With HushTalk we tried to handle communication in an object-oriented style i.e. by calling handler objects with events rather than calling callback functions. In our example we use HushTalk to set up communication between chat applications and for the communication between the browser and remote application.

3. Realization in Hush

In this section, the implementation aspects of our approach are discussed.

3.1. Hush

For the realization of our example, we used the Hush[Eliëns 1995] package for all components i.e. the browser, HushTalk and the chat application. Hush is a C++ API for Tcl/Tk and introduces some basic concepts such as a kit, a handler and a widget. The kit class is the abstraction for the graphical toolkit that is used, in our case Tcl/Tk. A handler is an object that is called when events occur that need processing. A widget is an item visible to a user, examples of widgets are a button and a scrollbar.

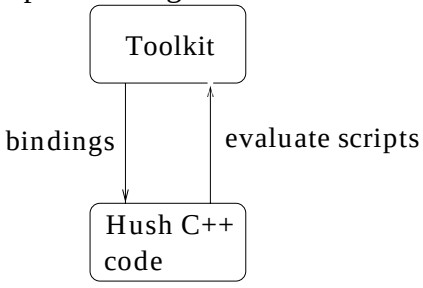

Figure 3: Relation between Hush C++ code and a Toolkit

For this paper the kit class is the most important one because it provides access to the underlying Tcl interpreter by the kit::eval(Tcl_String) method. This way Tcl code can be used on a C++ level. To get from a Tcl level to a C++ level, Hush allows bindings to be defined between Tcl commands and Hush handler objects. Whenever the Tcl command is called, the handler object will be activated with a Tcl event. This way a two way relation between Hush code and a Toolkit is established.

3.2. Creating new widgets

One of the features of Hush is that new widgets can be added. These can be accessed via both a Tcl and a C++ interface. The chat application of our example HTML page is also a new widget. In our approach, any application that is written as a new widget can be used. So the Tcl script that is contained in the chat page consists of only two lines :

```
chat .c
pack .c
```

Since chat is not a standard Tcl widget, not every interpreter will support a chat widget unless it has been installed. The interpreter used by our browser does not have a chat widget either. It would not be wise to link every interesting widget with the browser, leading to a large monolithic browser. We employ an other solution. When an applet tag is encountered, the value of the class attribute is interpreted as the name of a new widget and a dummy widget is installed in the interpreter. This way, any widget can be used in an HTML page. The widget actually becomes a remote widget since the real widget implementation may reside elsewhere. In the next section will be explained how remote widgets work as if they were locally installed.

3.3. Separating interface and program code

Hush applications use a Tcl interpreter to build the GUI of the application and the Tcl code that is produced during the lifetime of an application essentially forms the interface of the application. So to run an application on some host and have the interface in a browser involves separation of the interface and the rest of the application.

To realize this we developed new kit classes to do the work. Every Hush program needs at least one kit and each kit object has one interpreter that is accessed with the eval() method. For the remote application we created a server kit and for the browser a client kit. The client and server kit send each other messages concerning the interface or user interactions.

For instance, when the remote application calls eval(“entry .e”) to create an entry box, the server kit sends the Tcl string to the client kit in a request for execution. After the client kit has received the request and executed it, the result is sent back to the server kit. See fig 4. So now all the code for the interface is sent to the browser and evaluated there.

WWW browser	Chat application
	(object calls eval("entry. e"))
	(server kit sends request)
(client kit receives request)	
entry .e	
(client kit fetches result)	
(sends reply with result)	
	(server kit receives result)
	(return result to object)

Figure 4: Doing an eval()

Since a user can interact with the interface, user events also need to be sent to the remote application. Bindings that have been defined by the remote application are also reported to the client kit so it knows which user actions are to be sent back. For instance, a binding on a <RETURN> key could have been defined with :

bind .e <RETURN> { mycommand Return}

When a user presses the return key in the browser, the client kit delegates the key-press to the server kit which in turn will call the appropriate handler object with a Tcl event. See fig 5.

WWW browser	Chat application
(user presses RETURN)	
mycommand {return}	
(kit delegates command)	
	(receive command)
	(eval "mycommand {return}")
	mycommand {return}
	(handler object is called)

Figure 5: Delegating user interaction

So now the two-way relation has been re-established. We have separated the interface and the application but to the user there’s no difference except for some performance loss.

By designing new kits for the client and the server, any application that uses such a kit can be used in a client-server fashion with a web browser. Since every Hush program needs a kit, all that is needed is to recompile the program with a server kit.

4. Evaluation

In this section, our approach will be discussed and compared to other possible approaches.

4.1. Related work

A similar example such as our chat page could also be written as a Java applet. However, there would be some important differences to our approach. Writing a chat applet in Java would require the use of sockets for sending data to an other applet e.g. an other browser. Apart from the security risk of doing this, sophisticated application (or applet) management will be difficult to realize when all the work has to be done by the applet itself. The ToolTalk session is a central external process that efficiently takes care of the management job.

Since all the code for a Java applet is in the page itself, the applet is executed locally instead of remotely as in our case. So no communication between the applet and the browser would exist. A consequence is that if an applet is executed on a remote host, other services or data on that host can also be accessed by the applet. This make a new range of applets possible. However it could also be seen as a drawback since the local file system of the browser cannot be used for security reasons.

A last difference is that since only code for building the interface of an applet is sent to a browser, the rest of the code of the applet does not have to be public and sent to the browser. This is also a big advantage for more commercial purposes. In other words, we employ mobile interfaces instead of mobile code.

4.2. Security

Security has become an important topic now that more and more pages contain applets that are executed locally. How secure our approach is depends on how secure the components are. A remote application only sends information due to eval(Tcl_string) or the creation of bindings. A browser cannot control a remote applications since it only sends Tcl results or delegates bindings. However, a remote application can control a browser by sending Tcl strings that would have sensitive browser-data as a result. Therefore we use safe interpreters in the browser to execute the scripts in. The safe interpreter does not allow file I/O. Although it solves the security problem it also makes it impossible for an applet to store a file for the user on his file system. A more elaborate security scheme is desired to fix both problems.

Finally, another security problem is caused by ToolTalk itself. ToolTalk does not offer any security whatsoever, so any application that somehow manages to join the right session, could receive the messages that are sent. One solution could be to encrypt the data in the messages. So our approach is not safe as it is now but does not introduce any new security holes.

4.3. Some considerations

Although our example shows that integrating applications and browsers is possible, there are some points of considerations. These points need to be addressed for future development of the concept.

- **Multiple views**

  When a browser connects to the remote kit there can be only one client kit with one server kit. It would be nice to have multiple clients connect to one server so there would be multiple views on one widget. But that leads to problems with the semantics of the eval() method. The eval() method returns a string containing the result of the Tcl command. But if the command was sent to multiple client kits, the server kit would receive multiple answers that may be very different. This leads to messy semantics for the result of Tcl commands. One solution can be to restrict the eval() method so it does not return a result. This would lead to a situation where the clients can control the state of the remote widget but the remote widget can not control the state of the client.

- **Starting applications**

  In our example the remote application has to be started by the ``service provider'' since the remote application runs under the user identity of the remote user. It is possible to compile a static message pattern into the ToolTalk system so that ToolTalk automatically starts the application when there is a message for it. But if some other user would request the same chat demo page, the chat application will already be running and the new client will use it as well. If clients would act as multiple views, this would be an acceptable situation but would not be if there must be a one to one correspondence between clients and servers. Clearly a mechanism is needed to distinguish between these situations. This problem probably makes ToolTalk less suitable.

- **Addressing a remote widget**

  In our example the remote widget is addressed via an X server name and the widget name. But if there were more identical remote applications running on that X server it will be ambiguous which remote application will be used. There is no mechanism to address a specific remote application. This is typically something that is needed when there are multiple viewers and multiple remote applications, for instance when editing multiple music scores with several people at the same time. ToolTalk offers possibilities to send messages to specific processes identified by a process identifier but it is only available when the application has been started. A solution requires that an application can be addressed uniquely whether it is running or not.

Solutions for the above points are not plausible. It seems that ToolTalk may not be suitable for the task by lack of needed features. For future versions we are considering a CORBA[OMG 1991] implementation of HushTalk. A possibility that also needs to be looked at, is how the functionality of a WWW server would have to be extended to fulfill the role ToolTalk now has in our system. It could easily start applications on the same host and adapt document contents concerning the applets.

5. Conclusions

Our approach shows that applications and web browsers can be integrated to offer functionality that otherwise can not be obtained using ordinary web browsers and

servers. This new functionality is needed for cooperative work since the HTTP protocol lacks the necessary functionality. By using HushTalk as an other communication protocol on top of HTTP, we are able to provide basic support for cooperative work on the web in particular on session management. As a result, our approach makes use of mobile interfaces instead of mobile code as the Java approach does. We think that the combination of mobile interfaces and a new communication protocol tuned for cooperative work is a promising route for integrating CSCW and the web.

6. References

Eliëns A (1995), Hush - a C++ API for Tcl/Tk. In: The X Resource, Issue 14, April 1995

van Doorn M and Eliëns A (1995), Integrating applications and the World Wide Web. In: Proceedings of the Third International World-Wide Web Conference April 1995

Gosling J and McGilton H (1995), The Java Language Environment: A White Paper. Sun Microsystems 1995

van Ossenbruggen JR and Eliëns A (1995), Bringing Music to the Web. In: Proceedings of the Fourth International World Wide Web Conference. December 1995

Julienne AM and Holtz B (1994), ToolTalk & Open Protocols: Inter-Application Communication. SunSoft Press/Prentice Hall, 1994. ISBN 0-13-031055-7

Eliëns A, van Ossenbruggen JR and Schönhage SPC (1996), Animating the Web -- an SGML-based approach, International Conference on 3D and Multimedia on the Internet, WWW & Networks, Bradford, 17-18 April 1996, British Computer Society

Object Management Group (1991), The Common Object Request Broker: Architecture and Specification. Revision 1.1,1991

Ousterhout JK (1994), Tcl and the Tk Toolkit. Addison-Wesley Publishing 1994