Distributed Applications in a Hypermedia Setting

Krishna BHARAT

Graphics, Visualization & Usability Center
Georgia Institute of Technology
Atlanta, GA 30332, USA
Email: kb@cc.gatech.edu

Luca CARDELLI

Digital, Systems Research Center
130 Lytton Avenue
Palo Alto, CA 94301, USA
Email: luca@src.dec.com

ABSTRACT

This paper addresses the issues involved in embedding distributed applications in a hypermedia web. In particular, we describe how applications generated in the Visual Obliq programming environment were integrated with the World Wide Web. We validate our design decisions by addressing pragmatic issues relating to the authoring process, security, heterogeneity, and response time, and by analyzing the activities involved in launching and joining a distributed session.

1. INTRODUCTION

Hypermedia creates a pseudo-world where interconnected documents may be shared by a community of users. Recently, our view of what constitutes a document has broadened to include more than traditional “passive” media-types. Much of the recent interest in the World Wide Web [Berners-Lee 92] has been fueled by its potential for embedding online services. Already, interactive forms for issuing queries and sending messages, and “electronic cash” mechanisms for paying for services, are becoming available in hypermedia documents.

In collaborative work, multi-user applications create “sessions” that are micro-cosms in their own right. Each session is comprised of users, possibly in different parts of the world, who sporadically communicate and exchange information. It would only seem natural to merge the micro-worlds formed by individual multi-user sessions into the larger context of a hypermedia web.

In this paper we show how the four major activities required to support multi-user sessions, namely:

- Finding out what applications are available,
- Starting an application, and hence launching a session,
- Making a session publicly known, and
- Joining a session,

may *all* be mapped to actions performed on a hypermedia web. Specifically, we show how the Visual Obliq distributed environment (described in [Bharat 94]) was integrated with the World Wide Web (often known as just “the web”).

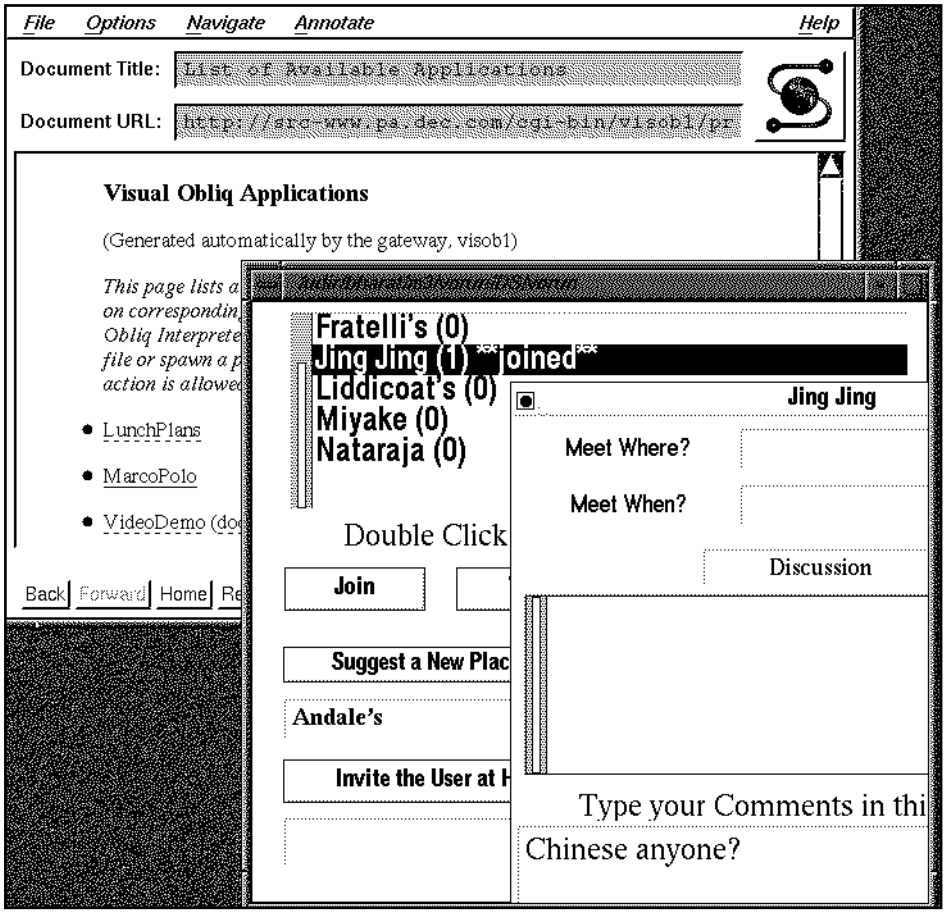

Figure 1: The LunchPlans application deployed from within NCSA’s Mosaic

In the resulting environment, the user is first presented with a web document (usually called a “page”), which lists a set of applications in Visual Obliq. Each application-name is highlighted to show that it is “active.” As shown in Figure 1, when the user clicks on an application’s name (e.g. LunchPlans), its code will be brought over and run locally.

The code that is retrieved from the web is executed on the user's machine, in a *safe* manner; safe in the sense that the application will be prevented from doing anything malicious or indiscreet in the user's environment. If it is a multi-user application, this action implicitly creates a session, which is automatically registered with the web. Specifically, the organization that made the application available maintains a dynamic list of sessions under way and the participants in each session.

Users who wish to join ongoing sessions can access a corresponding "Sessions Listing" document, which gets created on the fly and lists all sessions that are active at that time. As before, names in the listing have active links. Clicking on a session-name will bring over code that will connect the user to the session. Again, this will happen in a safe fashion.

It must be noted that the hypermedia environment merely acts as a mediator; it provides access to applications and connects users to sessions but does not get involved in the sessions themselves. Once a session is established, the different parties in the session no longer need the hypermedia environment to communicate.

2. VISUAL OBLIQ

Visual Obliq [Bharat 94] is an environment for designing, programming and running distributed, multi-user GUI applications. It consists of an interface builder for interactively constructing the application, and run-time support to handle the distribution. The interface builder was designed on the lines of Microsoft's Visual Basic, but specifically for distributed applications, and provides an integrated 'draw-program-and-go' solution. Moderately sophisticated distributed applications can be designed, programmed and tested in a matter of minutes. The video [Bharat 95] demonstrates the interactive construction of a multi-user editor with floor control, in under seven minutes.

The Visual Obliq interface builder outputs code in an interpreted language called Obliq [Cardelli 95], which inherently supports distributed computing. When a Visual Obliq application is started by the user at some site, it creates a "session." Other sites can join the session by acquiring a reference to the session called a "session-handle." They do not require the application program code. Once they connect to the session using the handle, the necessary procedures will be shipped to them over the network from the site that started the session.

There are many ways that a prospective participant may acquire a session-handle. One way is for the participant to use the names of the application, and the host where it was launched, to compute where the handle is stored. This address is then used to explicitly "import" the handle. This is the technique that we employ here.

3. IMPLEMENTATION

In the World Wide Web, organizations run servers to make their documents available over the network, while users run clients to fetch and display web documents. The clients and servers talk a protocol called HTTP. In our implementation we changed neither the server nor the client nor the protocol. Instead we used the extension mechanisms provided at the server and client ends to register the software support needed.

3.1 Document Provider's Perspective - Server Side

Any self-contained Visual Obliq application can be placed on the web. The interface builder is able bundle all its program code into a single file, suitable for transmission over the wire as a web document.

There are two kinds of hyperlinks in the web:

- **Direct links**, that cause the server to fetch the specified documents directly, and
- **Indirect links**, that cause the server to hand off the task of fetching documents to specialized programs called gateways.

Our server side extension was through a gateway program called "VO-Gateway," with the following duties to perform:

- When the user clicks on an indirect link to an *application*, VO-Gateway is invoked with the command `get-application` and the application-name as an argument. It then ships the text of the application as a document, having attached to it a MIME header, to give the document a type. The header shows that it is a Visual Obliq application. Further, if it is a distributed application, the gateway records the fact that a session has been started.
- When the user clicks on a indirect link to a *sessions-listing*, the gateway is invoked with the command `get-listing`, and communicates with each session on record to check if it is still alive. Finally it returns a document listing the names of sessions that are presently active, with links to the sessions.
- When the user clicks on a indirect link to a *session*, VO-Gateway is invoked with the command `join-session` and the session-name as argument. This causes it to generate Obliq code that will connect the client site to the session by importing the session-handle. As before the code is given a MIME header to show that it is a Visual Obliq application and shipped to the client.

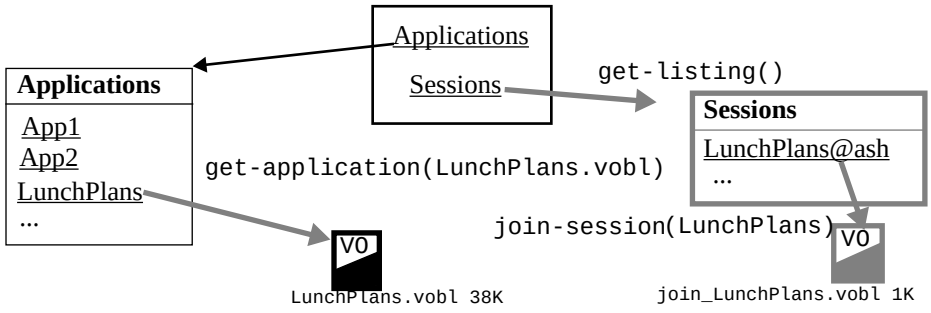

Figure 2: Web structure at the document provider's server. Indirect links and dynamically created documents are shown in gray. Each indirect link points to the document that is eventually returned by the VO-Gateway. Typical document sizes are shown to give an idea of relative access times. The icons labeled 'VO' represent Visual Obliq programs, augmented with a MIME header.

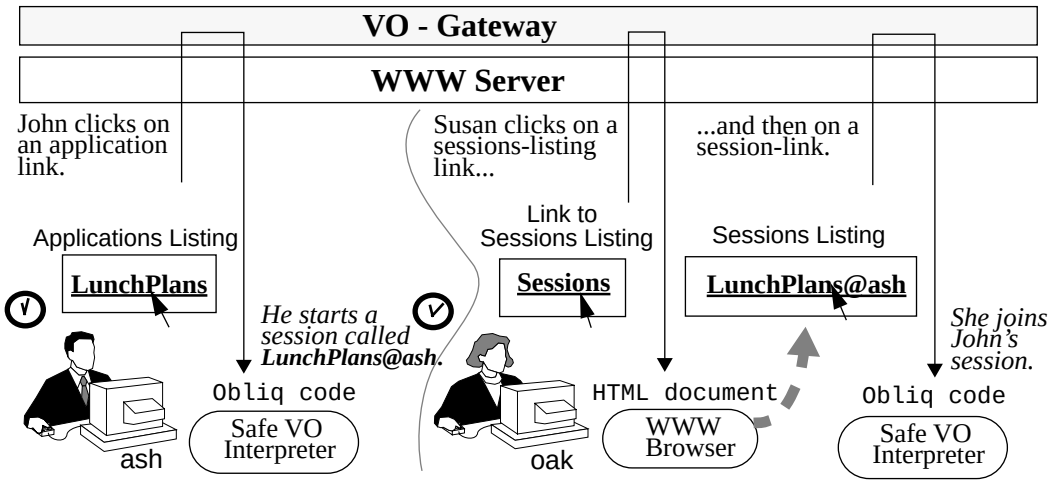

Figure 3: Bird's eye view of a distributed session

3.2 End User's Perspective - Client Side

At the client end, it is possible to register "viewers" to display documents with media types the WWW client is not able to display. For instance postscript documents may get handed off to a postscript viewer. For our purpose, we bound the MIME type corresponding to Visual Obliq applications to a "Safe" Visual Obliq interpreter. When the application program is received over HTTP, it is immediately channeled to the interpreter, and gets executed on the client's machine as if it were locally resident.

The Safe Visual Obliq Interpreter is a special-purpose Obliq interpreter used to interpret the applications that come over the network. In Obliq, all unsafe operations are readily identified by the fact that they require the use of "access" handles to system resources. This interpreter is considered safe because it conservatively screens all unsafe operations based on their arguments. As guidance it uses a user-specific configuration file containing regular expressions that specify which operations are to be allowed and which are to be blocked. Operations that are allowed by the configuration file are executed in the regular manner. When a blocked operation is encountered, the interpreter notifies the user that the program is attempting something illegal and aborts the application. When an unsafe operation falls in neither category, which is the default case when no preferences have been specified, the interpreter rewrites the operation in a human intelligible form, and pops up a notice to ask the user if it should be allowed to go through.

4. Design Rationale

The implementation that we have described supports well the four activities needed to launch and maintain distributed sessions (listed in Section 1). In each case, a single click on a web page is all that is required.

We now evaluate our design on the basis of several other pragmatic considerations:

- **Resonse Time** - The response time for clicking on each of the three types of indirect links (application, session-listing and session) is increased by the time required to invoke the VO-Gateway program. This adds about 5 seconds to the time required to retrieve the document. The code size of average Visual Obliq programs (38K for `LunchPlans.vobl` and less than 1K for the code to join `LunchPlans`) is of the same order as normal web documents. However, users are willing to tolerate much larger response times when launching an application (especially when it is remotely accessed), than when downloading passive documents. In the case of application and session links, there is the added overhead of starting the Visual Obliq interpreter at the client end, to parse and execute the incoming application, which is again a function of the size of the program. In the case of LunchPlans, the overall retrieval and launch time is around 40 seconds. This compares favourably with the several minutes needed to download, uncompress and possibly compile applications using conventional means. Other interpreted languages such as Perl and TCL have gained popularity on the web for the same reason. We are planning to precompile Obliq programs to byte-code to save the interpreter parsing time, thereby further reducing the response time.
- **Security** - Whenever executable code comes from outside the local environment, it poses a threat to security and privacy. Nonetheless, users frequently download and run “shareware” and other forms of code to be found in public repositories. Distributed applications, which are guaranteed to be “network friendly,” pose a larger threat than usual. Whenever an attack is mounted, it involves access to basic system resources such as the disk, operating system, network or the display. Regulating access to these resources is a reliable way to enforce security. Interpreted languages are attractive in this respect because interpreters provide the control needed to regulate access robustly. In Section 3.2 we described how the Safe Visual Obliq interpreter solves this problem. Other interpreted environments have similar “safe” interpreters. Currently we only regulate access to the disk and the operating system, and so our system does not guarantee privacy within the session. We are in the process of moving our implementation to a more “secure” transport layer, with encryption, to solve this very problem. It is worth noting that the restrictions we impose on incoming programs are not unreasonable, since such applications, which “come over the wire,” are expected to bring whatever resources they need with them, and should make minimal assumptions about the system resources at the client end. Notice that we use HTTP only to connect users to the session. All communication within the session may be conducted over safe, encrypted channels.
- **Heterogeneity / Interoperability** - Distributed applications are more demanding than others because they need to be run on multiple architectures and interoperate. The SRC Modula-3 distribution provides Visual Obliq interpreters on numerous platforms, which are intended to interoperate. A more subtle cause of interoperability is version mismatch. It is important that all parties in a distributed session make use of the same version of the application. Fortunately, in our scheme, the application is downloaded from the site where it is maintained, every time the session is started. Other users get their code directly from the initiator of the session. Hence incompatabilities are eliminated. This design elimi-

nates the inertia that prevents distributed applications from being frequently upgraded. To appreciate the advantage of this, consider for instance, the common UNIX `talk` facility. Although it is easy to suggest ways in which the `talk` protocol may be improved, it is extremely difficult to make any changes, because that would require changing implementations all over the world!

- **Ease of Authoring**- There are two aspects to authoring. First, the distributed application itself needs to be created. In [Bharat 94] and [Bharat 95], we illustrate how the Visual Obliq programming environment simplifies this process. Secondly, the application needs to be made publicly available over a hypermedia protocol such as HTTP. Figure 2 shows how this may be done in the context of the World Wide Web. To make an application available (e.g. `LunchPlans.vobl`), all that is needed is an indirect link of the form `get-application(LunchPlans.vobl)`. Similarly, a `get-listing()` link will make session listings available.
- **Ease of Use** - Users do not need a modified WWW browser to run sessions. All they need to do is modify their environment to register the Safe Visual Obliq interpreter as the default viewer for the MIME type corresponding to Visual Obliq applications. Also, to avoid being interrupted frequently, they need to set up a configuration file that specifies how unsafe operations are to be treated. Typically, this will be copied from a prototype created by the system administrator.
- **Robustness** - The designers of the World Wide Web took care to separate gateways from the server and viewers from the client, so that errors in documents do not cause the system to crash. Consequently, the distributed session cannot bring down a web server or client. If the server crashes on its own accord, it will still be able to link new users to the session upon restarting, since the sessions registry is persistent.

5. CONCLUSION

We believe that hypermedia is a natural environment to embed multi-user sessions. Groupware applications have traditionally implemented their own session creation and listing facility. With the increasing popularity of the World Wide Web, it seems like a more suitable location for such services. The web's HTTP protocol is able to subsume the communication needed to initiate, advertise and connect users to sessions. In this paper, having motivated the need for embedding distributed applications in a hypermedia web, we have shown how this may be done in practice, and listed many of the attractive features of our design.

6. REFERENCES

[Berners-Lee 92] Berners-Lee T., Cailliau R., Groff J., and Pollermann B., "World-Wide Web: The Information Universe", *Electronic Networking: Research, Applications and Policy, Vol. 2(1)*, pp. 52-58, 1992, Meckler Publishing, Westport, CT, USA

[Bharat 94] Bharat K., and Brown Marc H., “Building Distributed Multi-User Applications By Direct Manipulation”, *Proc. ACM Symposium on User Interfaces Software and Technology*, Marina Del Ray, CA, Nov 1994, pp. 71-82.

[Bharat 95] Bharat K., and Brown Marc H., “Building A Distributed Application Using Visual Obliq”, To appear in *CHI ‘95, Video Proceedings.*

[Cardelli 95] Cardelli L., “A Language with Distributed Scope”, *Proc. of the 22nd Annual ACM Symposium on Principles of Programming Languages*, Jan 1995, pp. 286-297.