Supporting the Construction of Distributed, Interoperative User Interface Applications

Krishna Bharat
Graphics, Visualization & Usability Center
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332
kb@cc.gatech.edu

ABSTRACT

Distributed applications are hard to construct. They are both hard to design and hard to implement. Part of the complexity is due to the lack of the right programming abstractions -- remote procedure calls and sockets are low level and too far removed from the needs of user interface developers to be useful. It is also in part, due to the lack of suitable application design tools in the distributed arena. Specifically, tools which integrate the various tasks involved in developing user interface applications, and also support the right structuring abstractions needed to build distributed applications.

The goal of this thesis is to: a) motivate the need for tools and abstractions that bridge the gap between the application programmer's needs and the facilities provided at the operating system/network level, b) provide paradigms for structuring distributed, interoperative UI applications, and c) show how these may be couched in an integrated application development environment, with abstractions that make programming and deploying a distributed application easy.

KEYWORDS: Programming environments, GUI builders, distributed applications, agent migration, migratory applications, distributed constraints, distributed algorithms, CSCW.

INTRODUCTION

Although the operating system and networking support for distributed computing has been around for several years now, we have yet to see a proliferation of multi-user, distributed applications. This is because building distributed applications is hard, and whenever possible, people tend to use pre-existing mechanisms such as *talk*, *nv*, *vat* and *e-mail,* instead of building their own custom tools. Often distributed applications do more than just help people communicate; they allow people to collaborate on a shared context, and hence can require arbitrary kinds of interaction and dependencies between application components. Hitherto, programmers have relied on message passing or remote procedure call style mechanisms to program such systems. These mechanisms are low level and too far removed from the needs of distributed UI application developers to be effective. Direct manipulation GUI builders have proven very successful in supporting and integrating the interface design, programming and testing phases of application development. However, the program structuring paradigms they employ do not scale to a distributed setting. To be truly effective the application builders themselves need to be aware of the distributed nature of the applications being built. There is clearly a need for tools, abstractions and programming paradigms to bridge the gap between what application developers need and what programming environments are able to provide.

DEVELOPMENT ENVIRONMENT SUPPORT

The Visual Obliq[1] application builder (Fig. 1) supports the 'Shared Scope' paradigm for the construction of distributed applications (described in the next section). The builder allows windows called 'forms' to be drawn and programmed. Each form is converted by the builder into a class of distributed objects that can be instantiated on multiple machines. The environment comes with distributed support for spreading the session and maintaining connectivity between sites.

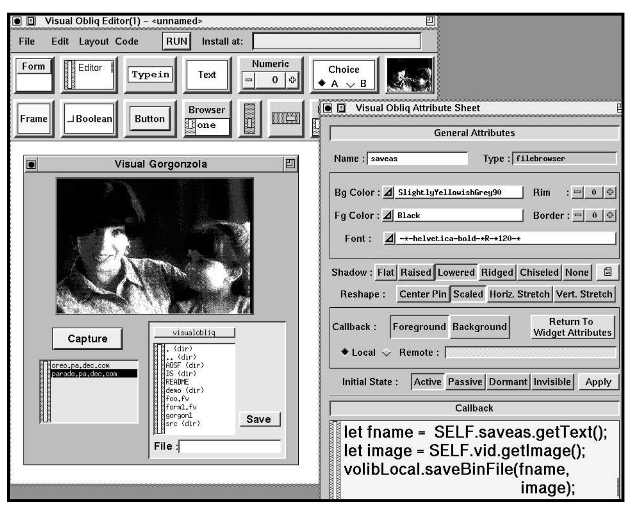

Figure 1: The Visual Obliq application builder

PROGRAMMING PARADIGMS

Next, we list a few programming paradigms that we have had the opportunity to implement and study. This list is by no means exhaustive. No single paradigm will suffice in all cases, and often a combination of approaches may be required.

The 'Shared Scope' Paradigm

In this programming model (supported in Visual Obliq[1]), the application starts at one site and spreads to other sites by transmitting some of its code and part of its scope. All sites share a portion of the application's initial scope, which contains global code and data, and implements a global name-space. The unit of distribution is a form (a programmable window object). Forms in other portions of the distributed application can be accessed via the global name-space, and operated upon transparently. The session spreads by the transmission of an abstraction called the 'session constructor' which brings over the necessary code to a new site, as well as the shared scope. The constructor then instantiates forms.

This approach was found to be well suited for launching and sustaining distributed sessions on the web (described in [2]).

The 'Bridged Scope' Paradigm

In the previous case, the application was required to evolve from a single site. In the 'Bridged Scope' case, the application transmits part of its computation and scope to a peer, to be executed within the confluence of the two scopes.

The 'Autonomous Agent' Paradigm

This is a variant of the 'Bridged Scope' approach. Here the transmitted computation brings over all the state and code that it needs, allowing it to break ties with the original site. This is done by bringing over a set of handles to the relevant application state and 'deep copying' all the needed data-structures and code over. In UI applications the agent will need to bring over windows as well, with their state intact. This poses an interesting problem in the context of windowing environments because much of the interface state is kept externally, in UI-libraries or in the window server. Thus the application needs a way to checkpoint its user interface state into application objects, destroy the windows, transmit its application state and rebuild the UI from the saved context.

The amount of state that is transmitted will decide how much of the application will move to the new site. In the limiting case, when all active windows migrate, the agent may be the entire application! This was the case in our implementation of 'Migratory Applications' in Visual Obliq[3].

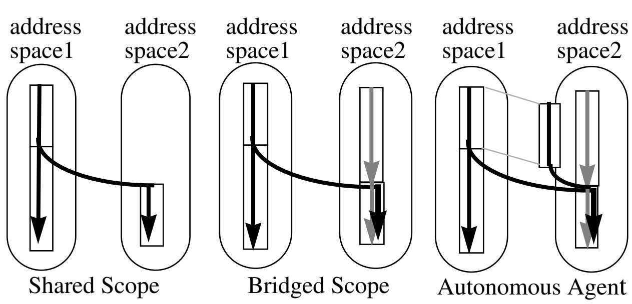

Figure 2: Various Paradigms based on Scope

Distributed Concurrent Constraints

This is radically different from the scope-based approaches described so far. In this paradigm, application code and data reside statically in individual address-spaces, but are connected by a graph of inter-address-space data constraints. The constraints allow a set of applications to execute cooperatively by maintaining a sequence of relationships between values situated in distributed contexts. For instance, this would allow changes in the data in one address-space to be reflected in graphical views in other address-spaces. While the use of constraints to maintain relationships *within* a user interface application has been well researched, the use of constraints to maintain links *across* applications is relatively new.

The hard problem (it turns out) is not merely building such constraint systems, but implementing them efficiently. This is because concurrent updates often tend to overlap and interfere, and the constraint graph is itself subject to change. Previous distributed approaches[5] have been synchronous, allowing no concurrency whatsoever. The distributed constraint graph is a major bottle-neck, and traditional constraint maintenance algorithms scale very poorly. To address this, we have developed a highly efficient, distributed algorithm for concurrent, one-way constraints, called Doppler[4]. The algorithm requires no global synchronization or locking. It tracks causal relationships between graph updates, and allows updates to proceed concurrently whenever possible. In terms of concurrency, the algorithm is provably optimal.

CONCLUSION

Currently, programming environments and paradigms for building distributed user interface applications are virtually non-existent. In my thesis, I have explored a few approaches to solving this problem, and described some that seem to work quite well. Almost all the work mentioned here has been implemented and published.

I wish to acknowledge the significant contributions of Marc H. Brown, Luca Cardelli, and my advisor, Scott E. Hudson. My expected date of graduation is June 1996.

REFERENCES

[1] Bharat, K. and Brown, M. H., "Building Distributed, Multi-User Applications by Direct Manipulation", *Proceeding of UIST '94*, Marina Del Rey, CA, Nov. 1994.

[2] Bharat, K. and Cardelli, L., "Distributed Applications in a Hypermedia Setting", *Proceedings of International Workshop on Hypermedia Design '95*, Montpelier, France, Jun. 1995.

[3] Bharat, K. and Cardelli, L., "Migratory Applications", To appear in *Proceedings of UIST '95*.

[4] Bharat, K. and Hudson, S.E., "Supporting Distributed, Concurrent, One-Way Constraints in User Interface Applications", To appear in *Proceedings of UIST '95*.

[5] Hill, R. D., "The Rendezvous Constraint Maintenance System", *Proceedings of UIST '93*, Atlanta, GA, Nov. 1993.