Action at a Distance: A Case Study in Composability

Sean W. O'Malley

Department of Computer Science
University of Arizona
Tucson, AZ 85721
Tel: (602) 621-6613
Email: sean@cs.arizona.edu

Abstract

We believe that the key to software reuse is the ability to arbitrarily compose a library of single purpose modules into large systems. Over the last five years we have created such a system (the x-kernel) which supports the arbitrary composition of protocols into large protocol graphs. We have designed a number of communications systems which extensively reused protocol modules and have constructed numerous small protocol modules which were designed from scratch to be reusable. This paper attempts to give some insight as to how such protocols are produced and the problems that can interfere with composition.

Keywords: module compostion, reuse pragmantics, reuse experience

Workshop Goals: The integrating of reuse theory and practice.

Working Groups: domain analysis, tools and environments, reuse and object oriented methods

1 Introduction

The x-kernel[1] supports a model of software reuse where collections of small single purpose communications protocols can be arbitrarily composed into novel communications systems. A communications system is represented as a directed acyclic graph of protocol objects. The edges of the graph explicitly define the uses and depends upon relationships between protocol objects. Objects may be multiply instantiated.

As our approach to modular protocol reuse has evolved there has been a constant problem with how to support interaction between two protocols arbitrarily separated in a DAG. We have called this the problem of action at a distance. Action at a distance is basically caused by protocol interactions which cannot be represented in our graphical notation. These iterations can radically reduce the potential composability of a collection of protocols and hence seriously inhibit reuse. In the worst case one ends up with a collection of protocols which can only be composed in one specific way. While our own experience with action at a distance is with communications protocols, we suspect that this is a common problem with reuse systems based upon arbitrary composition.

Action at a distance is commonly found when decomposing a monolithic protocol implementation into a collection of single purpose micro-protocols. For those protocols to be truly reusable, all action at a distance problems must be resolved in the most general way possible. If no such solution is found every change in the protocol graph will require code modification in the individual protocols. In this paper I will describe the action at a distance problems encountered while developing a set of modular RPC protocol components and describe several attempts to resolve these problems.

2 Sprite RPC and xSprite

The protocols described here are a result of decomposing the Sprite RPC protocol developed at Berkeley. A description of the basic Sprite RPC protocol can be found in [2]. Sprite RPC implements the basic Birrell-Nelson[3] remote procedure call protocol. A partial and simplified pictorial representation of our modular implementation of Sprite RPC (xSprite) is given in Figure 1. This graph provides a super-set of the semantics found in Sprite RPC but cannot interoperate with Sprite RPC. xSprite differs from Sprite RPC in that it can be used over the Internet and it supports very large messages. More complete descriptions of xSprite can be found in [4][5].

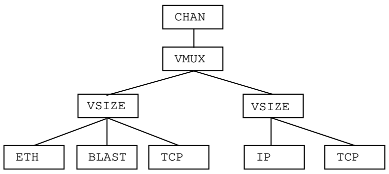

Figure 1: The xSprite Protocol Graph

xSprite consists of the following protocols CHAN, BLAST, VSIZE, and VMUX. xSprite uses the Ethernet and the standard Internet protocol suite to deliver packets. CHAN implements the basic Birrell-Nelson RPC mechanism. BLAST implements an optimistic blast algorithm to send large messages between hosts connected via a LAN. VSIZE dynamically selects a transport subgraph based upon the size of the outgoing message. VMUX statically selects a transport subgraph based on the address of the target machine. If the target machine is reachable on the local net the Internet protocols are bypassed. This graph uses the Ethernet (ETH) to send short packets locally, BLAST to send medium packets locally, TCP to send very large packets locally, IP to send short packets across the Internet, and TCP to send large packets over the Internet. The protocol graphs under all the transport protocols are ignored for simplicity.

3 Action at a Distance Problems

This decomposition of Sprite RPC created at least two action at a distance problems. First, for xSprite to implement the same semantics as Sprite RPC, BLAST cannot positively acknowledge the arrival of all the fragments of a large message. Thus the client BLAST does not know when to delete the fragments that were sent to the server. However since CHAN will get an acknowledgment for every RPC, it does know when BLAST can free its fragments. The question is how to get that information where it needs to go. The second action at a distance problem arises from the fact that ETH, IP, and BLAST are unreliable protocols but TCP is reliable. For both performance and correctness reasons CHAN should not retransmit the body of a message when it is using a reliable transport protocol.

4 Solutions

Originally we attempted to solve the first problem as follows. When a message is sent, BLAST would return a ticket to CHAN which would uniquely identify the storage associated with that message. BLAST supported a free resources operation which would free the message fragments associated with a valid ticket. When CHAN received an acknowledgment for a message it would invoke the BLAST free resource operation with the ticket. Unfortunately the only hint that CHAN had about the location of the BLAST protocol was that it was located below CHAN in the protocol graph. Thus the free resources control operation had to be inherited up the protocol graph. This is expensive and unfortunately will not work if there are two instances of BLAST configured below CHAN.

What is required is for some form of temporary communications link be set up between CHAN and whatever lower level protocol was used to send the current message. Hence the path of the message in question determines which lower level protocol needs to communicate with CHAN. Given that in the x-kernel messages traverse the graph using a series of successive xPush operations, the transport protocol could return any required information to CHAN as the return value for the xPush operation.

CHAN requires three pieces of information to resolve its action at a distance problems: an object pointer to the transport protocol it should invoke a free resource operation on, a ticket which allows the transport protocol to uniquely identify the message in question, and a reliability flag. By default CHAN assumes that there is no need to free resources and the underlying protocol is unreliable. Thus in the graph, ETH and IP do not have to return any information. BLAST must return a pointer to itself and a ticket. TCP need only return the information that it is reliable.

5 Recursion

While the x-kernel does not support cyclic protocol graphs there are instances of static recursion, for example composing an instance of the protocol BLAST on top of another instance of the protocol BLAST. This composition results in a protocol graph which can handle much larger messages than BLAST itself. Unfortunately as defined above there is no way for CHAN to free the resources of the lower level BLAST protocol. Hence BLAST must support the chaining of CHAN information requests. If BLAST receives a transport protocol and a ticket as the return value from one of its xPush calls it must save that information and perform a free resources operation on each saved transport protocol when CHAN performs a free resources operation on it. In addition since CHAN is a reliable transport protocol its xPush operation must return this information.

6 Conclusions

In any reuse system based upon the free composition of software components maintaining composability is a constant concern. The temptation to resolve action at a distance problems in an ad-hoc fashion inevitably leads to a reduction in the composability of the system as a whole. We have identified in the x-kernel a case where the communications required is defined by the subgraph traversed by a specific message rather than a static protocol graph. By treating this as a general problem a solution was derived which should restore arbitrary composition to a collection of important protocols. While every effort was made to insure generality the information exchanged may still be too CHAN specific. This approach should be tried on other protocol graphs. Work is also needed to come up a semantically cleaner way to implement these interactions.

7 References

References

[1] N. C. Hutchinson and L. L. Peterson, “The *x*-kernel: An architecture for implementing network protocols,” *IEEE Transactions on Software Engineering*, vol. 17, pp. 64–76, Jan. 1991.

[2] B. B. Welch, “The Sprite remote procedure call system,” Tech. Rep. UCB/CSD 86/302, University of California Berkeley, Berkeley, Calif., June 1988.

[3] A. Birrell and B. Nelson, “Implementing remote procedure calls,” *ACM Transactions on Computer Systems*, vol. 2, pp. 39–59, Feb. 1984.

[4] N. C. Hutchinson, L. L. Peterson, M. Abbott, and S. O’Malley, “RPC in the *x*-Kernel: Evaluating new design techniques,” in *Proceedings of the Twelfth ACM Symposium on Operating System Principles*, pp. 91–101, Dec. 1989.

[5] S. W. O’Malley and L. L. Peterson, “A dynamic network architecture,” *ACM Transactions on Computer Systems*, vol. 10, pp. 110–143, May 1992.

8 Biography

Sean W. O'Malley is an Assistant Research Scientist in the Department of Computer Science at the University of Arizona. He has been working on the x-kernel, a modular architecture for protocol development. He leads an effort in the development of a suite of modular and reusable security protocols. His work is funded by ARPA and the National Computer Security Center. Before that he was an Assistant Professor of Computer Science at the University of Texas at Austin. He received a Ph.D. in Computer Science from the University of Arizona in 1991.