Flexible Distributed Programming using Mobile Code

David Halls, John Bates and Jean Bacon
{David.Halls,John.Bates,Jean.Bacon}@cl.cam.ac.uk
University of Cambridge Computer Laboratory
Pembroke Street, Cambridge CB2 3QG, United Kingdom

Abstract

We have designed and implemented a mobile code system. It has been integrated with an existing distributed programming environment that includes a Remote Procedure Call (RPC) system, object interface definition language, a trader, an event system and continuous media streams. This bespoke platform for distributed programming has allowed us to experiment with mobile code and propose new facilities.

Section 1.1 discusses our implemented system. Section 1.2 describes a novel remote procedure call system built on top of our mobile code system. Related work is reviewed in section 2. Current and possible future uses of our system are given in section 3.

1 The *Tube*

1.1 Description of Implementation

We have implemented a new platform that is able to receive programs from a network and execute them. It is called the Tube and is portable across machine architectures. The programs it receives are written in Scheme and marshalled before they are sent. The marshalling respects duplicate objects and cyclic references.

Programs can thus be written which will run on any machine in a heterogeneous network of computers. We call an instance of a Tube platform a REP-site. The programs it runs are known as Remotely Executable Programs, or REPs, because they can be sent over a network to a remote REP-site for execution there.

REPs are given a special environment within which they can access only certain symbols. They execute in a safe interpreter and are only allowed to access those functions that the REP-site's owner allows them to.

Because of the code-data duality of Scheme, REPs can easily create data structures within themselves and then submit these as REPs to remote REP-sites or to their host REP-site.

We have implemented the Tube so that any Scheme data structure can be marshalled and sent as a REP. As well as ordinals, strings, lists and vectors, functions (lambda expressions) can be sent. They are sent along with their closing environment; the default behaviour is to copy all of the environment but variables can be forced to be references back to their originating site. Function marshalling is made possible by executing REPs under a continuation-passing metacircular evaluator. We also use this to capture the continuation (dynamic state) of a REP. Continuations can be marshalled too, so that a REP can be stopped, moved and restarted somewhere else. We can do much more than just execute byte-code compiled scripts remotely - REPs can modify themselves, create and dispatch other REPs and treat their functions and even state as first class, transmissible data.

There are a number of points to note regarding the internal structure of REP-sites. Firstly, REP-sites are multi-threaded. Full access to a POSIX threads interface provided by the underlying operating system is given to REPs. Many operating systems support POSIX threads either internally or via a user-level library.

Secondly, a REP-site itself consists of a number of threads providing services. Other threads (typically REPs) contract services from them via internal communications queues as part of a computation. REPs themselves can set up as service providers.

Thirdly, a noticeboard is provided, which a REP can post messages to or read messages from. Data values can thus be left by a REP for others that may arrive later to use. The noticeboard can be divided into different areas; an access control list is associated with each.

Finally, REP-sites can dynamically load at run-time compiled libraries of code and call functions contained in them. This allows facilities to be added to a REP-site without having to stop and recompile it.

Some value-adding features have been added to the Tube system, that utilise the facilities described above.

A user interface toolkit has been integrated with Tube REP-sites. Functions to create and manipulate widgets are available to REPs. We have also enhanced the toolkit with the ability to return the state of any arbitrary user interface and the widgets on it as a series of bytes. A corollary function takes a saved user interface and recreates it in a visible form.

This allows a REP to create a form on one machine and retain it as embedded state when it moves to another. As much as possible of each widget's state is remembered.

We have also written a Netscape plug-in that allows the Tube to be used for writing applets which may be referred to from inside World-Wide Web Pages. Pre-marshalled REPs are saved in files and referred to from HTML. Netscape reads the REP and passes it to a REP-site. The REP is executed at the REP-site and displays its user interface in the window within the HTML page that Netscape makes for it. This architecture separates the browser from the mobile code system. Unlike monolithic approaches (such as a Java interpreter linked with the Netscape executable), we are able to plug in different browsers or even different applet languages when we develop them.

The Tube has also been integrated with our existing distributed programming environment. REPs are able to make RPCs using our IDL-based RPC system, lookup object instances in our trader, register for and receive events from our event system [BHLM95] and control the configuration of named streams of multimedia data between active objects. We are thus able to use the Tube as a command language for instantiating and invoking methods on objects. This gives a scripted layer of control which can be used to build higher-level specifications of systems. We are able to use REPs written in the scripting language to configure parts of applications and move them around. Because configurations of groups of objects as application parts are written as scripts, we can perform domain-specific re-configuration of them when they move. Using the Tube allows us to integrate new types of active objects when they are written.

1.2 Tube-RPC

A separate, novel RPC system is built on top of the Tube itself. In general, distributed applications built using RPC consist of a number of clients and servers that communicate with each other using pre-defined and fixed interfaces. Clients are linked in with compiled communication stubs generated from the interface definitions. When clients use stubs that are compiled once from a definition to communicate directly with a server, some constraints are imposed on the server. If clients are not to fail (or perform marshalling incorrectly), the server must neither change the signatures nor remove any of the procedures it offers. Whilst the server can impose versioning and force clients to rebind, neither approach gives the client information about exactly what the server is doing.

Moving (some part of) the server into the client allows it to give information about any changes made in service provision through up-calls in the client's environment, facilitating server-driven per-client adaptation that takes into account client state. This also allows for per-client optimisation of communication with a server. Supporting plain-text reference by clients of server procedures at run-time allows a server to change its interface without having to worry about the underlying RPC mechanism breaking. When this is combined with a system of negotiation between client and server over the (declarative) naming of services on offer then changes in service provision could provoke a client to re-negotiate its interface with the server.

This re-negotiation introduces a dynamic interface between the two parties and represents an advertisement published by a server for clients to take hints from. By relaxing the requirement that a client contracts a service with a fixed interface, more flexibility can be achieved. However, it does break the notion of fixed classes of communication between clients and servers.

Tube RPC supports this mode of operation by allowing an RPC server to send clients a closure which, when run, installs itself there and acts as the server's proxy in the client. The client is freed from having to know how to contact and communicate directly with the server - it conducts all dialogue with the proxy. This allows the server to define at run-time what it offers to clients: it can update the implementation of its proxies at any time, by downloading code to each client, to add or change facilities.

There is work to be done in the interface between clients and the server proxies. Clients must be written so that they are able to adapt as the proxy informs them of new circumstances. The Tube RPC mechanism is now in place, and we will experiment with changing interfaces and shifting the balance between client and server. As well as coping with typing problems, we will ensure that clients which want to use a simple mode of operation that does not require downloading of proxies can do so. We will implement some real applications using this technique and cope with auditing interface change through chains of clients and servers. We are proposing Tube RPC as one possible abstraction over mobile code; our experiments are at a very early stage.

A further extension is to replace the publishing of server addresses in a trader with the publication of small pieces of code; instead of a client looking up the address of a server and then explicitly binding to it, it downloads the code from the trader and executes it. The code is responsible for contacting the server and retrieving the proxy that the client can use in further communication. This abstracts particular methods of contact away from clients. Servers publish methods of contact and then give clients tailored proxies for communication.

This allows one easily to change the initial arrangement of clients and servers. At one extreme, the complete server functionality could be published in the trader so that clients would actually download and install servers locally. At the other extreme, the code published in the trader would be a simple network connection to the real server executing somewhere else. Subsequently, by using downloadable proxies, applications can dynamically be rearranged – for instance, a server could download (replicate) itself to all its clients and then terminate.

2 Related Work

There is a growing body of work on mobile code. Some of the latest efforts are discussed here.

Java The Java Virtual Machine (VM) [Gos95] has been ported to a number of architectures. Programs compiled down into VM bytecodes can be executed anywhere. The only language that currently compiles to this bytecode at present is Java [AG96]. Java VMs are typically found inside WWW browsers, where Java bytecode is downloaded as applets for execution there. The disadvantages with Java are that programs cannot themselves create other programs, embedding arbitrary data structures, and send them to other VMs for execution; and that a Java program cannot be stopped mid-flight and shipped elsewhere.

(Visual) Obliq, Distributed Active Objects Obliq [Car94] is a higher-order distributed language that allows immutable data including functions to be transmitted over a network. Mutable data is transmitted as references to the originating site. Qbliq does not support arbitrary first-class, transmissible continuations of running code; programs can however create functions themselves as explicit continuations for transmission or have the current continuation migrate implicitly.

One development of Obliq, Visual Obliq, supports movement of user interfaces too. However,

> Migratory multi-user applications (IV) are significantly more complicated to implement, since connectivity needs to be maintained as the migration happens. We have yet to tackle this class of applications. [BC95]

We have used the Tube to do exactly this (see section 3). Another development of Obliq also has begun to addresses this point. The Distributed Active Objects project aims to provide high-level support to make it "easy to write collaborative and distributed applications" [NB96]. They use active objects in Obliq to support transparent distribution. We use the Tube to support user mobility to move objects and reconfigure them when they reach their new destination.

Kali Scheme Kali Scheme [CJK95] is very similar to the Tube. It allows functions and continuations to be transmitted over a network. It adds distributed garbage collection but does not interface with a user interface toolkit at present nor a conventional distributed programming environment with RPC. We hope to gain direct development experience with Kali Scheme very soon.

Telescript Telescript [Whi94] is a commercial mobile agent system. It is has been in existence longer than most and allows a program to move or duplicate itself to another computer. Telescript has been designed from the start to support the mobile agent programming paradigm. It does have user interface support but appears not to support first-class, transmissible (anonymous) functions and continuations that can be constructed dynamically.

TACOMA The TACOMA project [JvRS95] is concentrating on operating system support for mobile agents. It supports agents written in a number of programming languages. This is a quite a large project and we have not looked at the code in detail. However, it does appear to concentrate on facilities that are useful in supporting mobile agents that move around and as such should be applicable in enhancing any mobile code system, including the Tube.

CyberAgent CyberAgent [Sof96] is a commercial mobile agent system built on top of Java. Instead of downloading Java applets, CyberAgent allows Java programs to be sent out into the network. It is not clear from the publicly available documentation whether their Java programs can create arbitrary functions on-the-fly for transmission. This is doubtful due to the lack of anonymous and interpretive closure generation in the Java language and the requirement to compile into bytecode from a file. The ability to capture and send continuations is also likely to be missing.

Dreme Dreme [Fuc93] is a distributed version of Scheme. All language constructs, including closures and continuations, can be moved around a network. Dreme is interfaced with a user interface toolkit and has distributed garbage collection. As such, it provides the same functionality as the Tube. However, work on Dreme seems to have come to a halt, whereas we plan to deploy the Tube in a number of application areas.

Mobile Service Agents This project aims to support ubiquitous access to information by mobile users:

> MSAs are self-contained pieces of software that can move between computers on a network. Agents can serve as local representatives for remote services, provide interactive access to data they accompany, and carry out tasks for a mobile user temporarily disconnected from the network. Agents also provide a means for the set of software available to a user to change dynamically according to the user's needs and interests. [Kna95]

It uses a modified form of Facile [TLP$^+$93], which provides a model of concurrent processes based on CCS and allows both functions and continuations to be moved around. User interface facilities are provided and some useful applications have been developed. This project shows what a high-level approach to distributed programming incorporating mobile code can achieve.

Active Networks The Active Networks project aims to allow network users "to inject customized programs into the nodes of the network. An extreme case, in which we are most interested, replaces packets with 'capsules' – program fragments that are executed at each network router/switch they traverse" [TW96]. The use of sending small program fragments around is:

> Active architectures permit a massive increase in the sophistication of the computation that is performed within the network. They will enable new applications, especially those based on application-specific multicast, information fusion, and other services that leverage network-based computation and storage. Furthermore, they will accelerate the pace of innovation by decoupling network services from the underlying hardware and allowing new services to be loaded into the infrastructure on demand. [TW96]

This project is currently concentrating on the design of such a system. It demonstrates that use of code that moves is being proposed from low-level networks to high-level distributed collaborative applications.

3 Uses and Future Applications

The Tube was first used for experimenting with REPs that traverse a number of nodes in order to deliver and collect events. The system was at an early stage of development and this first application involved the transfer of drawing events for the lazy update of shared drawing spaces. Whilst no further use of REPs to deliver events is planned (because of the relatively high latency involved in marshalling and interpretation), this approach has evolved into the use of REPs to configure different sites for client-specific event generation and notification.

The Tube has extended our existing distributed system with support for a scripting language. This allows us to prototype object implementations, object interfaces and their calling semantics.

In the future, we wish further to integrate the Tube with a traditional RPC system, in particular to allow Tube RPC objects to be given interface definitions so that other, third party components can access their methods. We would then be able to snapshot at any time the interfaces of Tube RPC objects into a series of IDL files. A system could be prototyped until the balance between client and server elements of each component was correct and then snapshots of their interfaces taken in order to gain a high-level view. At the same time we must experiment with Tube RPC itself in order to gain experience with the problems forseen in section 1.2. We believe it to be a general and powerful mechanism that we intend to exploit in any applications involving the Tube in order to evaluate its mode of operation.

We have also started to investigate the use of mobile code in Computer-Supported Collaborative Working (CSCW) applications. Mobile management entity objects are used to group a user's objects into those participating in particular instances of an application. When interfaced with an Active Badge event system, the objects are able to follow a user from room to room. In combination with visible components inside an integrated user interface, experiments with mobility-aware CSCW are possible and we have implemented such a system [BHB96, Bat96]. We hope to investigate this further in the near future alongside other novel uses of mobile code in a distributed setting.

Acknowledgements

We acknowledge the UK EPSRC for supporting this work through a studentship and under grant GR/J42007. Dr Bates is grateful for the support of Michael and Morven Heller and St Catharine's college, Cambridge. We are grateful to ICL for general support of our research group.

References

[AG96] Ken Arnold and James Gosling. *The Java Programming Language*. Addison-Wesley, 1996.

[Bat96] John Bates. A Framework to Support Large-Scale Active Applications. In *Proc. ACM SIGOPS European Workshop*, Connemarra, Republic of Ireland, September 1996.

[BC95] Krishna Bharat and Luca Cardelli. Migratory Applications. In *Proc. ACM Symposium on User Interfaces Software and Technology*, Pittsburgh, USA, November 1995.

[BHB96] John Bates, David Halls, and Jean Bacon. A Framework to Support Mobile Users of Multimedia Applications. *Submitted to ACM Mobile Networks and Nomadic Applications (NOMAD)*, 1996.

[BHLM95] J. Bacon, R. Hayton, S. L. Lo, and K. Moody. Using Events to build Distributed Applications. In *Proc IEEE SDNE Services in Distributed and Networked Environments*, pages 148–155, Whistler, British Columbia, June 1995.

[Car94] Luca Cardelli. Obliq: A Language with Distributed Scope. *URL* `http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-122.html`, June 1994.

[CJK95] Henry Cejtin, Suresh Jagannathan, and Richard Kelsey. Higher-Order Distributed Objects. *ACM Transactions on Programming Languages and Systems*, 17(5):704–739, September 1995.

[Fuc93] Matthew Fuchs. Dreme, for Life in the Net, September 1993. Author's email address: `fuchs@cs.nyu.edu`.

[Gos95] James Gosling. Java Intermediate Bytecodes. *ACM SIGPLAN Notices*, 30(3):111–118, 22 January 1995. Proceedings, ACM SIGPLAN Workshop on Intermediate Representations (IR '95).

[JvRS95] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. Supporting Agents in a Heterogeneous, Faulty and Insecure Network. In *Abstract, Work in Progress Session, 15th ACM SOSP*, Copper Mountain, Colarado, USA, December 1995.

[Kna95] Frederick Colville Knabe. *Language Support for Mobile Agents*. PhD thesis, School of Computer Science, Carnegie Mellon University, December 1995.

[NB96] Mark A. Najork and Marc H. Brown. Distributed Active Objects. In *Proceedings of the Fifth World-Wide Web Conference*, Paris, France, May 1996.

[Sof96] FTP Software. CyberAgent Home Page. *URL* `http://www.ftp.com/cyberagents/`, 1996.

[TLP+93] Bent Thomsen, Lone Leth, Sanjiva Prasad, Tsung-Min Kuo, André Kramer, Fritz Knabe, and Alessandro Giacalone. Facile Antigua Release Programming Guide. Technical Report ECRC-93-20, European Computer-Industry Research Centre, December 1993.

[TW96] David L. Tennenhouse and David J. Wetherall. Towards an Active Network Architecture. *To appear in ACM Computer Communication Review*, 1996.

[Whi94] James E. White. Telescript Technology: The Foundation for the Electronic Marketplace. Technical report, General Magic, Inc., 1994. (General Magic White Paper).