Indirect Calls: Remote Invocations on loosely coupled Systems

Carlos Baquero*

Distributed Systems - DI
Universidade do Minho - Braga, Portugal

February 29, 1996

Abstract

Integration of Mobile computers into Worldwide networks is traditionally managed by hosting them into the fixed network. We argue that this approach excludes some important forms of interaction. We present a communication mechanism, suitable for developing applications that take advantage of transient connections. These communications can be supported by several transport mechanisms, including Email and plain file copying between non networked computers.

1 Introduction

Future applications can be expected to follow two major trends, Mobility and Worldwideness. These two issues are strongly interrelated as Worldwide systems often experience disconnected operation under network partitions and Mobile computers are partially integrated (possibly hosted) into worldwide networks.

A common scenario for the integration of mobile hosts (MHs) is based on a fixed worldwide network to which MHs connect from time to time, either by fixed or wireless links. Initial work under this scenario resulted in a number of proposals for communication support [1]. We argue that this scenario misses important cases that are driven by the heterogeneity of machines and network configurations. Some of the concerns are:

- How to communicate with applications on machines with no network support ?
- How to handle communication between two applications that seldom run in the same period of time ?
- Could casual interconnections among hosts (mobile and fixed), driven by physical mobility, be used to deliver messages among some applications ?

These cases are not uncommon and can be expected to be more frequent in the future. Some users often proceed to daily moves of their mobile computers from the office to home, and may have a fixed host at home. In this case, at a given time the MH, if not isolated, can be connected to the office machines or to the home one. A way of handling communication among applications on all this machines would be desirable.

A common case is that of users with MHs meeting while disconnected from the fixed network and establishing a transient link. These casual or planned links could be used to foster communication among visited machines and networks, in a epidemic fashion [8, 3, 2].

In this paper we describe a mechanism for handling communication in a loosely coupled

*cbm@di.uminho.pt

environment. This mechanism can be used to program applications that tolerate a communication pattern based on one way messages with potentially long delivery delays. Example applications are Appointment Book managers [5], Bibliographic database synchronization [7] and sharing of WWW Bookmarks.

2 Remote call environment

The devised communication mechanism was intended to be orthogonal to the language stub mechanisms as long as some form of one way calls (with no return value) were supported by the chosen system. The Inter-language Unification (ILU) project [6] and tools, developed in the Xerox Parc, provide support for RPC like calls within a Object Oriented environment. These tools are based on a Interface Specification Language (ISL) and, from the parsing of a ISL specification, can produce client and server stubs for several languages, including C, C++, Modula-3, Phyton and Lisp. Since ILU, among other advantages, supported the specification of one way calls on the ISL, it was chosen as the application interface. ILU uses TCP channels for handling invocations. Our system will intercept these channels and create persistent representations of the invocations.

2.1 Client-Server programming under ILU

ILU tools generate client and server stubs from a ISL specification, in a way resembling `rpcgen` for the RPC environment. We briefly sketch the programming environment for the C++ language.

Considering that, for instancem the ISL describes an object with two methods, the generated server stubs will define a class that includes two abstract methods that represent the ISL methods. These methods include an appropriate signature for the C++ language and should be implemented in a subclass by the programmer of the server. Once define, the server code should register the object and enter a main loop in the server stubs. In the ILU 1.8 version, registering an object in a server leads to the creation of a file in a `bindings` directory in the file system. This directory should be made accessible to the clients and contains information that identifies the object and the dynamic TCP port used to communicate with it. The main loop in the server stubs ensures that registered objects handle incoming TCP connections.

Client code should include the stubs that define a proxy for the server objects and after making a *lookup* of a published object, can use it with the generated method representations in C++. The lookup operation, as expected, will refer to the `bindings` directory.

2.2 Programming with asynchronous calls

Our application test case was a simple WWW bookmark (Hot List) synchronizer. This application was structured in two executables, `HotDaemon` and `HotClient`. The `HotDaemon` keeps the state of the common list and, would invocate other running copies of it in other machines, when a new bookmark as been inserted. As such this executable had both client and server stubs. The `HotClient` is executed with a bookmark as parameter, and will call its associated server, passing him the bookmark. Consequently, `HotClient` executables only include the client stub. A simple PERL executable can be used to parse the Netscape bookmarks file and make the necessary invocations to `HotClient`.

The defined ISL, for this test case, is as simple as possible.

```
INTERFACE HotWWW;
```

```
TYPE STRING = SEQUENCE OF SHORT CHARACTER;
TYPE URL = STRING;

TYPE HotList = OBJECT
  METHODS
    ASYNCHRONOUS AddURL (IN url : URL) ,
    ASYNCHRONOUS AskAddURL (IN url : URL) ,
  END;
```

`AddURL` is invoked by peer servers and will add the bookmark to the called server. `AskAddURL` is called from the clients adding the bookmark on the called server and leading it to issue the appropriate `AddURL` calls on the other servers.

3 Partitioning

Once the application is constructed and running in a fully connected environment, the ILU TCP communication and bindings can be intercepted, by our system, to provide indirect communication capabilities.

3.1 Interception and Delivery

Two daemons apply the necessary modifications to the existent communication protocol. The daemon, `Intercept-Freeze`, running on the client will intercept outgoing invocations, creating persistent messages in a `/tmp/out` directory in the filesystem. Another daemon, `Melt-Deliver`, running on the server will monitor a `/tmp/in` directory and deliver messages, when appropriate, to the application server daemons.

This process can be set up once the `bindings` directory is written by the server stub. With the binding information in this directory and once the directories cease to be shared, `Intercept-Freeze` will modify the bindings so that he will be called by the client stubs. Upon this calls, messages are written with file names that fully identify their destination. The file contents is the (opac) raw data produced by the ILU client marshalers.

Once the persistent packets reach the `/tmp/in` directory in the destination machine, `Melt-Deliver` opens TCP streams to the application server stubs delivering the raw data. Packets not relevant to this machine are just copied into `/tmp/out`.

Routing of messages from `/tmp/out` in a machine to `/tmp/in` in another machines can be done by hand, or with the aid of a PERL router that interfaces with several transport mechanisms, such as, emailing of uuencode files and use of RCP or FTP. The possibility of routing by hand is relevant to the use floppy disks or flash PC cards to foster communication between non networked machines.

4 Discussion

The option of using mobile computers or plain storage media to convey one way invocations among applications raises the important problem of how to control the delivery and the order of delivery of messages. Under this framework, messages are handed through replication among potentially good carriers, depending on their destination applications.

Traditionally, it would be highly desirable to ensure that messages from an application A to an application B, arrive at B in the order that they were issued in A. Unfortunately, this seems to be incompatible with the use of this class of useful but uncontrolled carriers. Although persistent messages are named in a way that identifies the order in which they were issued, this is (currently) only used to create unique names.

If incoming messages where to be delayed in the server machine until all relevant older messages arrived [4], there could be an infinite wait if a given carrier fails to make contact. This happens if we assume that not all ongoing messages are replicated in all carriers. One could argue that if the clients are notified of which messages were delivered,

then they could supply each carrier with all undelivered messages, so that when one carrier makes contact it can supply a whole sequence of ordered messages. Unfortunately, and again due to the uncontrolled behavior of carriers, this could fastly lead to unmanageable loads of messages if delivery notifications suffer long delays.

Due to these reasons the current system does not try to enforce a order on the delivery of messages. We believe that under some restrictions a ordered delivery of messages could be obtained. If so it should be presented as an alternative to the more flexible unordered delivery. These tradeoffs resemble those existent on TCP vs UDP communication.

The used instrumentation of the ILU system provided a rich and heterogeneous programming environment. Due to its orthogonality, with the consequent lack of modifications to the ILU system, it should be able to suit newer versions of the ILU system [1].

Naturally, large penalties can be expected on the delivery of invocations. These penalties should be understood in relation to the nature of the relevant applications. The support of indirect communication means that while some messages may take less than 1 second, others can take days, but still fulfill their objective.

Future applications that fit in this communication policy will be able to benefit from a mix of strong predefined interactions and casual unplanned interactions driven by user mobility in heterogeneous environments. Interaction on stable networks is easily supported by the use of RCP, NFS, and other of-the-shelf file sharing mechanisms. Email delivery can also enable interaction between machines with email capabilities but no TCP/IP connectivity.

[1] Apparently ILU 2.0 still supports the previous binding mechanism

5 Acknowledgments

Some of these issues were explored under the GIM project, that also included Jose Orlando Pereira, Antonio Luis Sousa and Rui Oliveira. The author would like to thank his PhD supervisor, Francisco Moura, as well as Vitor Guedes for the conversations around this work.

References

[1] B. Badrinath, A. Bakre, Tomasz Imielinsky, and R. Mrantz. Handling mobile clients: A case for indirect interaction. In *Fourth Workshop on Workstation Operating Systems*, October 1993.

[2] Carlos Baquero. Synergetic state evolution under mobile computing. Technical report, Distributed Systems, MInho University, 1995.

[3] Andrew Birrel, Roy Levin, Roger Needham, and Michael Schroeder. Gravepine: An exercise in distributed computing. *Communications of the ACM*, 25(4), April 19982.

[4] Silvano Maffeis, Walter Bischofberger, and Kai-Uwe Matzel. A generic multicast transport service to support disconnected operation. Technical report, Department of Computer Science, Cornell University, 1995.

[5] Jose Orlando Pereira and Antonio Luis Sousa. Group information manager. at http://luizinho.di.uminho.pt/gim/, 1995. Distributed Systems Group, Minho University.

[6] Mike
Spreitzer and et al. Inter-language unification. at ftp://ftp.parc.xerox.com/pub/ilu/ilu.html.

[7] Douglas Terry, Marvin Theimer, Karin Petersen, Alan Demers, Mike Spreitzer, and Carl Hauser. Managing update conflicts in bayou a weakly connected replicated storage system. In *Symposium on Operating Systems Principles*, Copper Mountain Resort, Colorado, Dec 1995. ACM.

[8] Marvin Theimer, Alan Demers, Karin Peterson, Mike Spreitzer, Douglas Terry, and Brent Welch. Dealing with tentative data values in desconnected work groups. Technical report, Computer Science Laboratory, Xerox PARC, 1995.