Support for Nomadism in a Global Environment

Position Paper

Bruce Jacob and Trevor Mudge
Advanced Computer Architecture Lab
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor
{blj,tnm}@eecs.umich.edu

Abstract. The goal of nomadic computing transcends simply making one's environment portable; mobile users require the ability to communicate with local servers despite location and to obtain local services despite statically defined service interfaces. To this end, we expect the portable computer or PDA to perform as a "universal interactor" [Theimer93].

The current paradigm for distributed computing, RPC, inhibits such free-form interaction. RPC requires static knowledge of the service and its interface; a programmer must know the interface and explicitly write code to use it. It is impossible, using present RPC implementations such as DCE or Sun RPC, to discover and use new services unless they conform exactly to the interfaces expected by the client.

We propose a new standard to define a fundamental level of support for nomadism—that of *service discovery*. It is the ability for a client to discover services based on descriptive names, bind dynamically to the servers that offer them, and communicate intelligently with the servers. In the full paper, we describe an implementation of service discovery that uses *descriptive lookup* and *dynamic interfaces*.

Introduction

Imagine the non-existence of *man -k* as well as the *-help* option that most utilities support. One would be forced to remember both the name and calling conventions of every program that one would ever need. We are similarly tying our own hands by using paradigms of distributed computing based on static interfaces, such as in Remote Procedure Call.

A fanciful scenario. You are mobile. You exit the plane and head to baggage. At the terminal, you respond to a page; your partner needs a fax of the changes you made in-flight. Your watch shows roughly an hour before your presentation. You connect to the airport's LAN and type

```
fax  postscript  Document=/u/blj/presentation.3a.ps  FaxNumber=3135551212
```

Your mobile client broadcasts for a directory object on the net. Luckily, one exists. The client sends a service inquiry to the directory object, asking for all matches on "fax postscript." The following list of matching service advertisements is returned.

```
name:LaserPrinter
addr:lpr@ohare.net
desc:laser printer, terminal B room 234a, $0.25 per page
keys:300dpi, postscript, ps, ascii

name:FacsimilePrinter
addr:faxserver@ohare.net
desc:prints incoming faxes from PSTN as well as LAN, terminal B room 234b
keys:fax, facsimile

name:FaxDoc
addr:faxserver@ohare.net
desc:faxes translated image of Document to DestFaxNo
keys:postscript, ps, ascii, gif, jpeg
```

Only one choice matches both "fax" and "postscript," so the client system sends a service inquiry regarding the *FaxDoc* service to *faxserver@ohare.net*. After a few protocol exchanges, the document and destination fax number are sent to the faxserver. A minute passes and your partner, still on the line, says, "Hey there. The fax just came in. It looks really good."

Motivation: a perceived weakness in the state of the art. What is wrong with this picture, besides the ease with which you "connect to the airport's LAN" and the omission of payment for the long-distance fax call? Using the prevalent paradigm of distributed computing, RPC (remote procedure call, [Birrell84]), this scenario cannot happen[1]. Two things occur that RPC does not

1. Clearly, this specific example could be implemented with a network *man* server, *rsh*, and simple shell redirection. One can certainly involve the user in simple service invocations that require such information as can be typed on the command line. However, there are more complicated scenarios which involve many client-server interactions. It would be cumbersome to involve the user in every instance of message-passing. A complex example demonstrating such a scenario, as in any sort of multi-phase reliable transaction [Pitoura94], group membership negotiation, or quality of service arbitration, would only serve to confuse at this point.

willingly support:

1. the lookup of a service based on a description of its capabilities (*descriptive* or *attribute-based lookup*), and
2. the dynamic interpretation of the service interface (*dynamic interfaces*).

At the moment you walked off the plane, your computer had never heard about *faxserver@ohare.net*, or the *FaxDoc* service that *faxserver@ohare.net* provides. In particular, nowhere in the code on the portable machine does a reference to any *FaxDoc()* procedure call exist.

By definition, RPC involves a procedure call. The procedure call makes an abstraction of and hides the entire remote service. However, to use the service behind the procedure call, it must be called, and more importantly, it must be explicitly coded into the program. This is the behavior of *static interface* systems such as RPC; a service's interfaces must be known not only before the transaction begins, but must have been known and used by the programmer at the time he/she wrote the client program.

This paradigm of computing fails to support *service discovery*: the act of finding new offered services with unknown interfaces, and using those services immediately.

It *will* become a problem. The rapid commercialization of the Internet and the National Information Infrastructure [Vernon94] imply that the future of general-purpose computing combines heterogeneity and ubiquitous computing services [Press94]. Future systems must not only support but expect a distributed environment similar to a shopping mall, where virtually all computing needs can be met. We have already witnessed the arrival of diverse user-level applications, from information services [BL92] to ordering pizza [Hut]. The de facto standard of distributed information services is the World-Wide Web [WebConf94], primarily because of the ease with which anyone can publish information.

In an environment where anyone can simply "hang a shingle" on the net and begin offering services, everyone will. Anything one can imagine and far more that one cannot will be available. In this environment, a mobile user will frequently encounter unfamiliar services, many of which the user will want to obtain. One should only need to know the description of a service to use it; this implies an amount of adaptability in the client machine. Client systems must be able to take advantage of new services that suddenly become available, even if the designers of the system did not foresee such services, i.e. even if the client has no *stub* for such a service. The system must not be restricted by static interfaces as in typical RPC mechanisms; it must be possible for a client to invoke a service knowing only a descriptive service name, using interfaces determined dynamically.

Where RPC fails

A large-scale nomadic environment requires service discovery. Similarly, any large-scale environment *without* mobility, but in which users can make inter-organizational service requests, also requires service discovery. Users will frequently come in contact with newly discovered services requiring interfaces unknown to the client at compile time. This happens in both environments, but is most easily visualized in a mobile setting. This section describes the needs of mobile computing and demonstrates where the RPC paradigm falls short.

The needs of nomadic computing. The following are scenarios demonstrating a mobile user's use of interoperability.

- At a presentation. Overhead slide projectors are used increasingly often to project a backlit view of one's portable screen. This implies much about the ease and reliability of developing and presenting a talk on the same machine. A conference room offering a large-screen display service would take display instructions in a variety of forms (X, display PostScript, etc.), and print them on the screen. Either every such display server must conform to the same standard invocation and setup interface, or the client machine must learn the interfaces dynamically.
- At a conference. Similarly, attendees at a conference might wish to display the current presentation on their own laptops, especially if the screen contains data or detailed illustrations.
- In a hotel. A hotel will likely offer services to patrons such as FaxDoc described earlier, or laser printing, or perhaps compute servers for guests with underpowered laptops. Again, either every service need conform to the same set of standard interfaces, or the responsibility for learning new interfaces lies on the client.
- In transit (planes, trains, and automobiles). Transportation services will likely begin to offer more communications services, much as airlines now offer in-flight telephone service. Likely examples would be information services and network/format translation (such as FaxDoc).

There are two components necessary to the realization of these scenarios.

First, a client must be able to perform attribute-based lookup. A client moving into an unfamiliar environment will not know the names of local servers or services. It is unreasonable to expect otherwise. However, such a client might need to obtain local services. The minimum amount of global knowledge should be required. This implies a well-known service broker or directory, similar to the Interface Repository of CORBA [Group93a], the portmapper in Sun RPC [Microsystems], or resource location in Grapevine [Birrell82]. The client should be able to find whatever services it needs by querying the local directory with a descriptive name or set of attributes.

Second, a client must learn the service interfaces dynamically. Just as one should not expect the client to know the name or address of the service or provider, the client should not need to acquire the interface description beforehand. Since most interface description files (IDL files) contain the unique id representing the service interface, the client knows the service name if it knows the service interface; they are linked. A more realistic view assumes the client to be unfamiliar with the interface. A client should therefore be able to acquire and use service interfaces dynamically, and a server should be prepared to send out descriptions of its interfaces to inquiring clients.

How RPC supports distributed computing. RPC mechanisms, as found in DCE [Foundation91], NCS RPC [Kong90], Sun RPC [Microsystems], and the Mach system call interface [Draves89], generally require interface description files at the creation time of client and server binaries. These files represent shared knowledge between the client and server and specify the structures of the different message types to be passed between the two. This paradigm contrasts sharply with the needs of a mobile environment, which suggest no *a priori* shared knowledge. In particular, we look at how RPC-oriented systems handle attribute-based lookup and dynamic interfaces.

Attribute-based lookup: Attribute-based, capability-based, or descriptive lookup allows a client requiring a particular service to find that service based solely on a description of the service. Most systems require that a client know at least the name of the service. Many require the client know the name of the server, or the name of the machine on which the server resides.

For example, before a client can connect to a server in DCE, the client must know the following things:

- The service id—a generated unique id from which the type of service rendered cannot be deduced.
  This: "002FD6B8-17F7-1B74-BFA9-02608C2C83B2" is an example. One cannot learn the name of a service except by obtaining a copy of the interface file.
- The service interface—a collection of message types to be passed between the client and server, each containing the service interface id, a message type id, and message-specific data.
- Either the authentication id of the server, or a shared secret between the client and the server—in this case a unique pathname in the CDS Namespace. No more than one client-server pair may rendezvous at a given CDS shared secret.

Less restrictive but equally problematic, Sun's RPC mechanism requires a service provider to be on the same host as its portmapper. This arrangement compels every host to offer the same set of services, or a client to know beforehand what host to contact. These types of requirements work fine in a closed system where the clients and servers are under the same administration and likely written by the same programmer. However, as systems get large it is unreasonable to expect clients to have global knowledge.

Dynamic interfaces: As mentioned earlier, a client must know the service interface before the transaction begins. In this RPC is inflexible. By definition, RPC involves a procedure call; this in turn requires a programmer to first learn and understand the procedure call interface, then compile it into the program. Without redefining the use of variable names and types in RPC interface definitions or inserting an protocol interpreter between client and server, an RPC client cannot use a service for which it lacks an IDL file. We *can* redefine RPC to include such a mechanism, but then it is no longer RPC.

Alternatives

RPC is a popular and wide-spread paradigm of distributed computing. However, RPC does not provide for attribute-based service lookup or the use of dynamic interfaces, both fundamental to mobile computing. This section describes environments that do support those facilities.

Related work and proposed standards. Ravishankar describes the problem in [Chang90]. He states that the client/server model is "a dominant system structuring paradigm" due in part to its support for system scalability. However, "systems have largely tended to use the server interface that implements a service as a representation of the service itself." He argues that the benefits of an abstract model are undermined by binding too many concrete implementation details.

The problem with RPC, as well as virtually all distributed mechanisms in large-scale use, is that a client interested in abstract functionality is committed to knowing too much ahead of time. To use a service, the client program must have an explicit reference to a function call providing the desired service, and often an explicit reference to a remote server. Systems like Prospero [Neuman93] and Cygnus [Ravishankar88, Chang90, Chang91] address this problem: they provide attribute-based lookup of services. They recognize that binding servers to services and their interfaces undercuts the flexibility of distributed systems, and instead allow clients to search for services using descriptions of the services.

However, half of the problem remains—that of implicitly binding clients to service interfaces. This is a real issue, recognized by the OMG and the ITU. Their CORBA and ODP standards [Group93a, CCITT92] address the issue, but since they are proposed standards they suggest functionality and say nothing about implementation. The CORBA standard advocates an intermediary between objects, called the Object Request Broker, that can translate between protocols. The client object must still know what information to send to a server and what information to expect in return. The responsibility of learning service interfaces thus shifts from the client object. However, it only moves one level higher to the ORB, which now bears the burden of learning any new interfaces. This is only a short-term solution. The ODP standard specifies a trading function in which server objects export service offers to traders and client object import offers. The recommendation does not specify how clients and servers are to communicate.

Service discovery in a distributed environment: principles. These are the general principles behind our implementation of service discovery, in which a client looks up a service based on a descriptive name, then acquires and interprets the service's interface at time of invocation. We assume the existence of well-known directory objects. Servers register their service advertisements with the directory objects, and clients send them service inquiries. Take as an analogy the Yellow Pages telephone directory—a mobile traveler only needs to know a few descriptive keys ("films", "movies" or "theaters" to learn where a film is showing) and what the universal payphone sign looks like (where one finds payphones one also finds directories) to locate virtually any service.

A client interested in a service performs the following steps:

1. locate a directory object,
2. submit a service inquiry to the directory object to obtain a list of service names and providers,
3. select and connect to a server,
4. submit a service inquiry to the server to obtain the service interface description, and
5. interact with the server in the manner needed to execute the service.

Conclusion

A commercialized Internet means that ubiquitous services are not far behind. A mobile user, or any user making inter-organizational service requests, will come in contact with many unfamiliar services and will probably wish to use one of them at some point. The present distributed computing paradigms do not support the use of services for which the client has no interface description at compile time. All require a specific function call within the client, most require the client to know the unique name of the service desired, and many require the client to know the name of the server or the server's host machine. This *a priori* shared knowledge stands in the way of mobility and the development of wide-area transaction services. We propose the concept of *service discovery*, in which a client can look up a service based on a description of the service, and then execute the service using a dynamically integrated service interface, obtained directly from the server. If an environment supports service discovery, it supports mobility. *Java* has the potential to become such an environment, but it is unclear whether it can overcome its performance and security weaknesses. The full-length paper presents our implementation of an environment supporting service discovery.

References

[Birrell82] Andrew D Birrell, Roy Levin, Roger M Needham, and Michael D Schroeder. "Grapevine: An Exercise in Distributed Computing." *Communications of the ACM*, 25(4):260–274, April 1982.

[Birrell84] Andrew D Birrell and Bruce Jay Nelson. "Implementing Remote Procedure Calls." *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[BL92] Tim Berners-Lee, Robert Cailliau, and Jean-Francois Groff. "The World-Wide Web." *Computer Networks and ISDN Systems*, 25(4-5), November 1992.

[CCITT92] CCITT. *Draft Recommendation X.903: Basic Reference Model of Open Distributed Processing*. International Telegraph and Telephone Consultative Committee, 1992.

[Chang90] Rong Nickle Chang. *A Network Service Acquisition Mechanism for the Client/Service Model*. PhD thesis, University of Michigan, 1990.

[Chang91] Rong N Chang and Chinya V Ravishankar. "A service acquisition mechanism for the client/service model in Cygnus." Technical Report CSE-TR-84-91, University of Michigan, 1991.

[Draves89] Richard P Draves, Michael B Jones, and Mary R Thompson. "MIG–The Mach Interface Generator." Technical Report (CMU unpublished report), Carnegie Mellon University, July 1989. URL=ftp://mach.cs.cmu.edu/usr/mach/public/doc/unpublished/mig.ps.

[Foundation91] Open Software Foundation. *DCE Application Development Guide*. 1991.

[Group93] Object Management Group. *The Common Object Request Broker: Architecture and Specification, Rev 1.2*. December 1993. OMG Document Number 93-12-43.

[Hut] Pizza Hut. *URL=http://www.pizzahut.com*.

[Kong90] M Kong, T H Dineen, P J Leach, E A Martin, N W Mishkin, J N Pato, and G L Wyant. *Network Computing System Reference Manual*. Prentice-Hall, 1990.

[Microsystems] Sun Microsystems. *Sun RPC man pages – rpc, rpcinfo, rpcgen, portmap*.

[Neuman93] B Clifford Neuman, Steven Seger Augart, and Shantaprasad Upasani. "Using Prospero to support integrated location-independent computing." In *Proceedings of the USENIX Mobile & Location-Independent Computing Symposium*, August 1993.

[Pitoura94] Evaggelia Pitoura and Bharat Bhargava. "Revising transaction concepts for mobile computing." In *Proceedings of the 1994 Workshop on Mobile Computing Systems and Applications, URL=http://snapple.cs.washington.edu:600/mobile/mcsa94.html*, December 1994.

[Press94] Larry Press. "Commercialization of the Internet." *Communications of the ACM*, 37(11):17–21, November 1994.

[Ravishankar88] Chinya V Ravishankar and Rong N Chang. "An attribute-based service-request mechanism for heterogeneous distributed systems." Technical Report CSE-TR-08-88, University of Michigan, 1988.

[Theimer93] Marvin Theimer, Alan Demers, and Brent Welch. "Operating system issues for PDAs." In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, October 1993.

[Vernon94] Mark K Vernon, Edward D Lazowska, and Stewart D Personick, editors. *R&D for the NII: Technical Challenges*. Interuniversity Communications Council, Inc., February 1994.

[WebConf94] WebConf. "Special issue: Selected papers of the first World-Wide Web conference." *Computer Networks and ISDN Systems*, 27(2), November 1994.