The use of distributed objects and dynamic interfaces in a wide-area transaction environment

Bruce L Jacob
Advanced Computer Architecture Lab, EECS Department, University of Michigan, Ann Arbor, MI 48109-2122
blj@umich.edu

Abstract. Distributed systems are in transition from LAN-based closed designs such as academic proof-of-theory architectures and application-specific commercial products, towards WAN-based environments where service requests cross administrative boundaries increasingly often. In such an environment, it is necessary for a client to bind to a server, either directly or indirectly, by the service name alone. It is unrealistic to expect that a client have knowledge about a particular server *before* attempting to invoke a service, as in DCE, the Open Software Foundation's Remote Procedure Call (RPC) package. This requirement is an artifact of closed-system RPC architectures, where the identities of the machines involved are known *a priori*.

This paper presents a CORBA-compliant middleware protocol for transparent service lookup and invocation in a wide-area network. Key to the protocol are an intermediary directory object that maps service names to objects that provide the services, and the concept of a *dynamic interface*, whereby a client gains the ability to interact with a server at the time of binding, rather than at compile time. Together, they allow clients to bind to servers using only service names, and to bind to services for which the clients have no IDL file. These abilities will become necessities in a wide-area transaction environment.

Keywords: middleware, distributed systems, persistent objects, dynamic service interfaces, transparent method invocation, network services

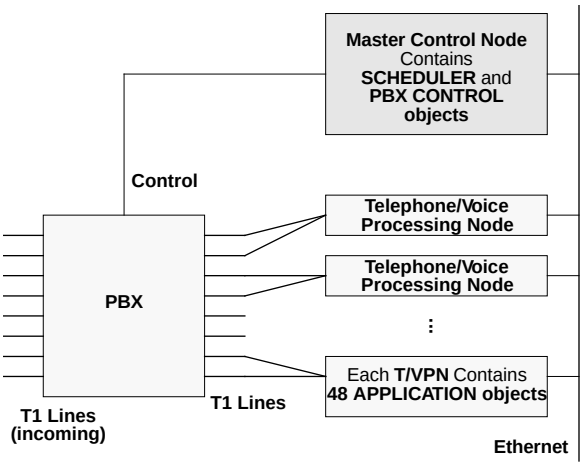
1.0 Introduction

The world is moving quickly to distributed systems, but the models from which systems are derived are often inappropriate. Many example systems and de facto standards are based on academic proofs-of-concept or commercial LAN-oriented products, both of which tend toward closed, tightly integrated implementations. These systems are not appropriate models when low-level service requests frequently cross administrative and organizational boundaries. They require that the client know too much static information about the server providing the service. This is often difficult to impossible to satisfy in a wide-area environment.

For example, most RPC systems, including OSF's DCE, Sun RPC, and Mach's system call interface [Foundation91, Microsystems, Draves89] require an RPC interface description language (IDL) file for the creation of a client *stub* before a client application may talk to a server. All communication with the server is through the stub: the stub handles the network connection as well as the marshalling and demarshalling of the data packets. In addition, the client must know how to reach the specific server: DCE requires the client know the principal id of the server (its authentication id), or a shared secret between the two (a location in the CDS namespace where the server can place binding information). 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 mechanisms are fine when the system in question spans a few machines administrated by one individual, but when clients expect to connect to servers on millions of machines, the model falls apart. For example, it is impossible to search for, discover, and bind to services for which one has no IDL file or server/host name.

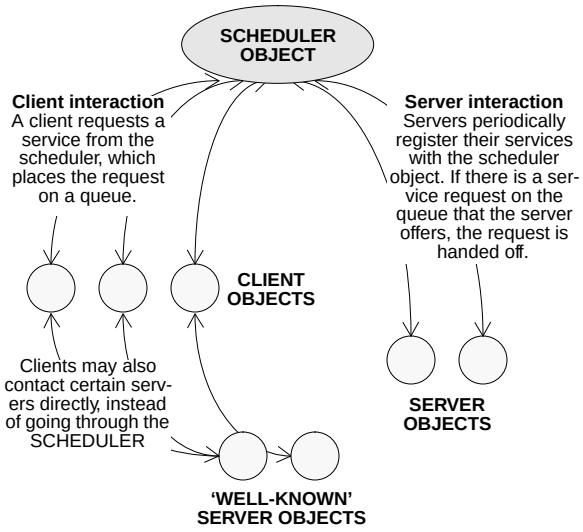
This paper presents the Distributed Services Environment, a middleware protocol for transparent service lookup and invocation in a wide-area network, based on a successful commercial telecommunications product. Both systems are the design and development work of the author.

The Telecom Services Architecture is a platform for commercial telecommunications applications, such as voicemail,



(a) Hardware Architecture

The architecture is centered around a "dumb" (host-controlled) telephone switch. Each Telephone/Voice Processing Node has 24 dedicated application objects listening to each T1 line (24 phone lines). When a call comes in, a service request is placed on the scheduler object's queue and a dedicated application object picks up the request from the scheduler. If the call requires specialized services, such as fax or speech recognition, the application object places an appropriate request on the scheduler's queue and the call is routed to a application object that can handle the request.



(b) Software Architecture

The software architecture allows clients to connect transparently to server objects, through a well-known server object called the scheduler. The object methods are invoked indirectly identified by the name of the service the method performs. A client can thus invoke a service by only knowing the name of the service, important as systems grow larger.

Figure 1. Hardware and Software of the Telecom Services Architecture

fax store & forward, automated paging, call connection, bulletin board services, and automated attendants (computerized receptionists). The system is a combination of specialized hardware and generalized software that handles a sustained call rate of more than 7 calls per second with bursts up to 40 calls per second; it is thus capable of handling over 25,000 busy-hour calls.

The software system design is independent of the hardware system. It is not designed specifically for telecommunications applications, but is a highly adaptable software platform for general distributed real-time applications. It consists of many cooperating persistent objects or agents. A centralized scheduler object takes requests from client objects and distributes them to server objects, much in the fashion of a job board. Client objects interface with the telephony hardware and register their services (object methods) with the scheduler so that services may be invoked transparently.

The Distributed Services Environment is an extension of this design. The analogy of a shopping mall can be used; clients need not know what servers offer the desired service, nor do they need to know the specifics of the interface, i.e. they do not need a copy of an IDL file to interact with a server. They need only know the address of a nearby 'mall directory,' which in response to a service inquiry will return a list of appropriate servers and descriptions of their offered services. The particulars of the exchange between client and chosen server are handled at the time of service invocation, through *dynamic interfaces*, as opposed to static interfaces such as those specified in an IDL file. The Environment is CORBA-compliant [Group93]; the details are beyond the scope of this abstract.

2.0 The Telecom Services Architecture

The hardware portion of the Telecom Services Architecture is pictured in Fig. 1a. It consists of a host-controlled digital telephone switch whose T1 lines are connected to application processing nodes that communicate with each other via Ethernet. The digital switch multiplexes T1 lines and is host-controlled (each T1 line carries 24 digital time-multiplexed individual phone lines, see [Tanenbaum89]). The processing nodes are composed of a CPU (iX86), memory, one or more SCSI disks, and telephone interface hardware.

The software portion of the architecture is pictured in Fig. 1b. Its design is orthogonal to the underlying telecommunications hardware. The system is based on the concept of distributed objects [Chase92, Jul88]. It is composed of trusted

persistent objects that fall into client and server categories. Well-known servers include a scheduler object, a database object, a mass storage object that handles the tracking of voice data, and a system event logging object, used for debugging. Clients include the switch control object and the telecommunications application objects, which also behave as servers when picking up service requests.

All objects are descended from a meta-class that offers a reliable UDP messaging service [Stevens90] as well as remote entry points into the well-known server objects. Objects do not reference each other's methods directly, but by the names of the offered services. This allows service invocations to be handled through an intermediary, so that objects need not know the identity of the object handling the service request.

To invoke a service, a client creates a service request message containing the identity of the service desired (a descriptive ASCII string) and hands the message to the scheduler object. The scheduler places the request on a timeout queue similar to the callout table in Unix. Server objects periodically register their services with the scheduler and if a registered service matches a queued request, that request is transferred to the server object. In the telecommunications product, the phone call is re-routed to the telephone switch port on which the server is listening. In theory, the transfer of control could entail anything.

3.0 The Distributed Services Environment

The Distributed Services Environment is based on the software architecture of the TSA. Client objects need only know the name of the desired service to connect to a server. The connections are made via an intermediary object similar to the TSA scheduler object. However, where the connection between client and server is made across the digital switch in the TSA, the client-server connection in the wide-area environment is a network connection initiated by the client. The intermediary object is a directory object that accepts a service inquiry and returns a list of appropriate servers and the services they offer. The client is free to choose among the server objects listed and contact any at will.

When the client object initiates direct contact with a server object, the client does not necessarily know the format or semantics of the server's messages. This service template information is given to the client by the server so that the client may interpret further messages received from the server. The arrangement constitutes a *dynamic interface*; the only information that a client *must* know is (1) the format of the *ServiceTemplate* message, and (2) how to use the information it contains to interpret further messages.

The protocol is at first glance similar to Sun RPC, wherein a client desiring an RPC service on a particular host contacts that host's portmapper, which passes the request to the appropriate server. However, there are three fundamental differences. First, the scheduler object provides access to servers on *any* host, not just the host on which the scheduler resides. This decouples the service from the host providing the service, a concept adopted in distributed file systems long ago, but slow to reach RPC systems. Second, the request is not handed off to the server directly, but instead the inquiring client is returned a list of viable server objects, and can make its own connections. This allows for tighter security than in Sun RPC, as each server object can handle its own client authentication. Finally, the client need not have a copy of an IDL file describing a service before contacting a server that handles the service. At invocation time, the client is given a protocol template by the server to be used in dynamic interpretation of messages. This allows clients to bind to new services as they come available, without requiring undue knowledge about the services.

References

[Chase92] Jeffrey S Chase, Henry M Levy, Edward D Lazowska, and Miche Baker-Harvey. "Lightweight shared objects in a 64-bit operating system." Technical Report 92-03-09, University of Washington, March 1992.

[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. 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.* December 1993. OMG Document Number 93.12.43.

[Jul88] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. "Fine-grained mobility in the Emerald system." *ACM Transactions on Computer Systems*, 6(1):109–133, February 1988.

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

[Stevens90] W Richard Stevens. *Unix Network Programming*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1990.

[Tanenbaum89] Andrew S Tanenbaum. *Computer Networks.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989.