A Survey of Remote Procedure Calls

B. H. Tay
A. L. Ananda

Department of Information Systems and Computer Science
National University of Singapore
Kent Ridge Crescent
Singapore 0511
BITNET: taybengh@nusdiscs

Abstract

The Remote Procedure Call (RPC) is a popular paradigm for inter-process communication (IPC) between processes in different computers across the network. It is widely used in various Distributed Systems. Although it is conceptually simple and straightforward to implement, there are a lot of different and subtle issues involved which result different RPC implementations. In this paper, various distinctive RPC implementations are surveyed, analyzed and compared: Xerox Courier RPC, Xerox Cedar RPC, Sun ONC/RPC, Apollo NCA/RPC, Cambridge Mayflower Project RPC, MIT Athena Project RPC, Stanford Modula/V RPC, and Rajdoot RPC are presented. The design objectives, features provided, call semantics, orphan treatment, binding, transport protocols supported, security/authentication, data representation and application programming interface of these RPCs are examined.

1. Introduction

Remote Procedure Call (RPC) is an easy and popular paradigm for developing distributed applications [26]. It is widely used as a communication mechanism in distributed systems, such as the Network Computing Architecture (NCA) from HP/Apollo [12] [13] [37] and Amoeba distributed operating system [32] [33]. Although RPC is conceptually simple and straightforward, there are a lot of subtle and difficult issues involved [23] [24] [30] [31]. Sometimes certain system parameters have to be traded off against other parameters in the design of different RPC. As a result, there are a lot of different RPC implementations available, both in the research and industrial environment.

The main objective of this paper is to select a few distinctive RPC implementations and give a brief comparison of them, their design emphasis, techniques, strengths and weaknesses. The

RPCs that will be discussed are the Xerox Courier RPC, Xerox Cedar RPC, Sun ONC/RPC, Apollo NCA/RPC, Cambridge Mayflower Project RPC, MIT Athena Project RPC, Stanford Modula/V RPC, and Rajdoot RPC. Note that this paper is different from Joshua Levy's paper [15]. Joshua Levy only examined the dependability and speed of the three commercial RPCs which include Sun, Apollo and Netwise RPCs.

There are other RPC implementations that are worth mentioning but not included here due to space constraint. These include the RPCs that are designed for certain special purposes in mind such as integration/accommodation of various RPC schemes and fault-tolerant. Heterogeneous Remote Procedure Call (HRPC) from University of Washington [5], MIT Argus [16] [17], Circus Replicated RPC [10] [11], Fault-Tolerant RPC [36], and CMU Multi RPC [20] [21] are excluded for the above reason.

2. Background

Before we present the comparison, let us clarify some of the terms used. In our definition, a blocking RPC is a RPC scheme which sends the request to server, and waits for result to be returned from server before returning back to application. A non-blocking RPC is a RPC scheme which sends the request and returns to application immediately without waiting for the returned result . Somehow the results can and will be retrieved later. Non-blocking RPC models closely to a message-passing protocol, except a remote procedure is explicitly invoked. This is probably an abuse of the original RPC design intention. Note that both definitions of blocking and non-blocking RPCs does not imply any call semantics. Either at-least-once or at-most-once, or even exactly-once call semantics can be implemented on both of them.

The second clarification we would like to make is on the call semantics. Currently, there are no agreed definition on the semantics of RPC. Hence we follow the semantics defined in Spector's paper [25] closely, except we term Only-Once-Type-1 as at-most-once and Only-Once-Type-2 as exactly-once.

2.1. Xerox Courier RPC

Xerox Courier RPC [35] is one of the earliest implementation of RPC. It is later referred by a lot of RPC designer to design and implement a new RPC. It was developed by Xerox as part of the Xerox Network System (XNS). It consists of three layer protocols: message stream, object stream and block stream. The middle layer protocol - object stream is roughly equivalent to a data representation standard. Courier RPC is the only RPC we examined here that solely relies on virtual-circuit for communication. The virtual circuit standard is defined by Xerox and is called XNS Sequenced Packet Protocol (SPP). According to Nelson paper [7], this approach of implementation produces unsatisfactory performance result.

Note that Xerox Courier is both a protocol and a specification language. The specification language is used to write RPC specification in order to generate both client and server RPC's stub code. Courier RPC is mainly used under Unix to communicate with other Xerox systems. For example, print servers or file servers.

2.2. Xerox Cedar RPC

Xerox Cedar RPC is also one of the earliest implementation of RPC. It was developed by Birrell and Nelson [6]. The development of Cedar RPC is regarded as one of the milestone in the RPC evolution.

The primary purpose in Cedar RPC is to make distributed computation easy. One of the main design principles in the Cedar RPC which affects several subsequent RPC designs is that the syntax and semantics of remote procedure calls should be as close as possible to those of local procedure calls. Also because of this decision, there is no timeout mechanism limiting the remote call duration (just like local procedure call) in the absence of machine or communication failure. Instead a probe packet was sent periodically from the caller (client) to the callee (server). Note that Cedar RPC was built on top of the connectionless-oriented transport protocol.

Both Xerox Courier and Cedar RPC provides a very good RPC model and implementation experiences for the later RPC developments. They both serve as the fore-runner in the RPC history.

2.3. Sun ONC/RPC

Sun ONC/RPC was developed by Sun Microsystems as part of the Open Network Computing (ONC). The other component defined in ONC is eXternal Data Representation (XDR) standard, which is also used by Sun RPC. The latest release of Sun RPC is version 4.0. Sun RPC [28] and XDR [27] specifications are available in Internet as Request for Comment (RFC).

Currently, Sun RPC provides both UDP and TCP for transport communication. At the same time, Sun RPC provides three types of RPC for each transport protocol. Thus after combining UDP and TCP, there are altogether six types of RPC calls. These RPC calls are blocking, batching and broadcast RPC. Batching RPC is like a non-blocking RPC call without expecting any return results, and the last call must be a normal blocking RPC call in order to flush all the calls. Therefore it is not considered as a true non-blocking RPC. In the broadcast RPC protocol, the servers that support broadcast only respond when the call is successfully completed, otherwise they are silent in the face of any failures or errors.

Sun RPC provides two types of interface for application programmers. One is available as library routines. The library routines consist of three layers, with the most difficult but most powerful routines at the lowest layer. The second interface is similar to other RPC implementations which uses RPC specification language (RPCL) and stub generator (RPCGEN). The RPCL is an extension of XDR specification.

Although there are complaints regarding the confusion between XDR and RPCL, Sun RPC model is one of the simplest model among all the RPC implementations. Moreover, it is quite autonomous (except relying on portmapper for binding) and flexible - provides both TCP and UDP, and 3 ways of naming mechanisms, i.e. via file /etc/hosts (BSD Unix), domain name server (DNS) or Yellow Pages (YP). Thus it can be easily ported to various operating systems and hardwares. Due to its simplicity, some of the programmers even straight code the

application using library routines instead of the RPCL interface. However, applications generated from RPCGEN (written using RPCL) are much easier to maintain where the stubs code are actually hidden. Hence the second interface is advised for application development.

The main shortcoming of Sun RPC is on its semantics. Since the call semantics are heavily depended on transport protocol, which creates inconsistency problem in the call semantics. For example, TCP based RPC is at-most-once, while UDP based RPC is at-least-once call semantics (although in the latest version 4.0, there is a cache maintained on server which attempts to achieve at-most-once semantics). In this case, the programmer must be aware of all these differences before coding any programs.

So far, the major application developed by Sun using Sun RPC is called Network File System (NFS), which is a very popular network file system. It was ported to many other platforms, including IBM PC (client side only) [3]. NFS is implemented using Sun UDP based RPC. It is carefully designed such that the operations performed are idempotent operations. The specification of NFS is also available in Internet as RFC format [29].

2.4. Apollo NCA/RPC

Apollo NCA/RPC was developed by HP/Apollo as part of the Network Computing Architecture (NCA) [12] [13] [37]. Apollo RPC is designed on top of various connectionless-oriented transport layers, which is opposite to Courier RPC. Currently, the datagram transport layers supported are DDS (Apollo proprietary protocol) and UDP.

The Apollo NCA is an object-oriented model, so is the Apollo RPC. One of the distinct features is that it uses a 128-bit, fixed length Universal Identifier (UUID) as the base identification mechanism for any entity on the network. The advantages of using UUID are location transparency, smaller size, the ease of embedding in the data structure, and the ability to layer various naming strategies on top of the primitive naming scheme.

Apollo NCA/RPC provides a rich set of RPC calls for programmer: a normal blocking RPC which is termed send-wait-reply (SAR) by Apollo, broadcast RPC, maybe RPC, and broadcast/may be RPC. Note that the maybe RPC does not expect any results back from server, and the last three types of RPC must be a request to an idempotent procedure in server. The maybe RPC can be classified as non-blocking RPC with no returned result. Although there are different RPC features provided, basically the call semantics supported can be classified into maybe, at-least-once and at-most-once semantics.

In addition of above call types, NCA/RPC provides extra routines which seldom found in other RPC implementations, i.e. the client is able to send "ping" packet to inquire about an outstanding request and "quit" packet to informs the server that it is to abort processing of the remote call. It also incorporates "call-back" (from server to client) mechanism to handle the case in which the server has executed the non-idempotent operation but fails to send the result, either because of network partitions or server crash. Thus, NCA/RPC is a reliable and dependable RPC, as concluded in [15].

Apollo RPC is tightly integrated with other components defined in the NCA. For instances, it relies on Conversation Manager to achieve at-most-once semantics and Location Broker for

binding. It is considered quite powerful due to its object-oriented nature, but the complicated and tightly integrated architecture which makes coding/porting difficult to some extent.

2.5. Cambridge Mayflower Project RPC

The Mayflower project group was set up in Cambridge to provide an environment for developing distributed applications and services [4]. It consists of a language - Concurrent CLU, a RPC communication protocol and an operating system - Mayflower supervisor. The Mayflower RPC is implemented on UDP (for Ethernet) and also on Basic Block Protocol (for Cambridge Ring).

The philosophy of Mayflower RPC is opposite to most of the other RPC approaches such as Cedar RPC. It emphasizes that the syntax and semantics of RPC should **not** be transparent to the programmer. As a result, the language was modified to add new syntax rather than using the method of stub generation often associated with most of the other RPCs. On the call semantics, the Mayflower RPC lets the programmer has a choice of semantics: MAYBE or EXACTLY ONCE. Note that the EXACTLY ONCE used in Mayflower RPC is actually equal to the at-most-once convention that we are using. It also provides exception notification to application. Mayflower RPC will signal the exception together with an error code to application whenever there is an error occurs during the execution of a remote call.

One of the distinctive features of Mayflower RPC is the extension of Concurrent CLU RPC to allow selection of its original defined data representation, XDR or Courier standard. Therefore the CLU clients are able to access existing Xerox and Sun services, and the Unix and Xerox Development Environment (XDE) and Interlisp clients can access CLU servers.

2.6. MIT Athena Project RPC

MIT Athena project [1] was embarked to integrate various computing and communication resources for educational purposes. Athena RPC [26] was developed as a prototype to evaluate how certain constraints influence the design and applicability of the RPC. The constrains are: no modification to the Unix kernel, port existing applications and support certain language suite.

Athena RPC provides blocking, and non-blocking call without returned result. Both of them were claimed to have at-most-once semantics, but actually the non-blocking call is only equal to the maybe semantics in Spector's paper [25] since there is no guarantee of delivery, nor are any errors or a result returned.

A request-response (UDP-RR) protocol is built on top of UDP in Athena RPC for RPC communication. The selection of UDP over TCP is mainly because of its less overhead (same reason as other RPCs). The Unix process model (heavy-weight process) posed a severe constraint to the implementation of Athena RPC on top of UDP. Since there is no light-weight process (thread) with share memory available in Unix, an Athena RPC server is actually implemented using two Unix processes and ports. One of the processes is responsible for performing the work, whereas the other process is in charge of responding to the keep-alive request. This results in a considerable performance and resource penalty.

2.7. Stanford Modula/V RPC

Modula/V RPC [2] was implemented in System V [8] and only supports Modula-2 programming language. The underlying transport protocol used is VMTP [7], a very fast, reliable transaction-oriented protocol. Giving the simplicity and speed of V and VMTP, it is not surprising that Modula/V RPC is one of the best performance RPC among the implementations.

Modula/V RPC supports both at-least-once and at-most-once call semantics, where at-least-once semantic is mainly used to perform idempotent operations. It then provides option to let programmer to specify whether an operation to be performed is idempotent or non-idempotent. Stating this option is a small price to pay compared to the gross efficiency gained.

In the Modula/V RPC, there is a team of process to handle a client request. The server (called dispatcher process) will signal another worker process within its team to handle a request if the request would cause the server to block. As a result, faster response time can be expected. Actually this is one of the advantages offered by V System, which provides some sort of light-weight process. However, there is a small problem associated to it, i.e. the V provides no efficient way for the worker process to reply directly to the original caller on behalf of the dispatcher process.

2.8. Rajdoot RPC

Notice all the RPCs mentioned above does not really have any orphan treatment, probably except for orphan detection but not killing. Instead most of the RPC implementations adopt orphans when it is detected. Rajdoot RPC [18] is specially designed to deal with orphan detection and killing. It is included to show the difference between those RPC with and without detection and killing.

Three mechanisms are employed in Rajdoot RPC, there are deadline, crashcount and terminator process. Deadline mechanism is the maximum time allowed for a server process to execute. Its purpose is to abort any potential orphans, thus preventing any future interferences. Crashcount is the number of times a node has crashed, which is a monotonic increasing value. Terminator process running on each node is responsible for terminating any orphans.

The main disadvantage of Rajdoot RPC is its deadline mechanism since it is very hard for application programmer to specify or predict a proper value.

3. Scoreboard

The comparison of various RPC implementations are listed in the table 1 and 2. Note that the RR in the table refers to request-response protocol, whereas the RRA refers to request-response-acknowledgement protocol.

	Courier RPC	*Cedar RPC*	*Sun RPC version 4.0*	*HP/Apollo NCA/RPC*
OS/NOS/DOS	BSD Unix	CEDAR/MESA	ONC/NFS	NCS
RPC Features	* Blocking	* Blocking	* Blocking * Batching * Broadcast	* Blocking * Maybe * Broadcast
Semantics	* At-most-once semantics	* At-most-once	* No Semantics (only infer from the underlying protocol)	*Maybe * At-least-once * At-most-once
Orphan Treatment	NO	* NO (Instead probe msg is used to detect the status of server)	NO	NO (Instead "Ping" pkt and Call-back are used)
Security/ Authentication	Xerox Authentication protocol: * simple * strong	* Grapvine DB * Encryption	* 3 different Authentication (NULL,UNIX,DES)	
Naming & Binding	* Xerox Clearinghouse	* Dynamic * Using Interface Name (Naming) plus Grapevine Database (location)	* Static (file) or Dynamic (YP or DNS) * Portmapper for locating a port * prog#, ver# -> port	* Location broker (use DRM-weak consistency) - Forwarding agent * Using UUID to identify entity
Stub Generator and Interface	* Courier interface and compiler (xnscourier)	* MESA interface modules definition and Lupine (Stub generator)	* RPCL (XDR extension) and RPCGEN compiler * Library routine (3 layers)	* NIDL and NCS/NIDL compiler
Data Representation	* Object stream		* XDR	* NDR
RPC Exchange Protocol	* RR	* Tailored RR	* RR	* RRA
Transport Protocol/Interfa -ce	* XNS SPP * BSD socket interface	* Datagram	* TCP/IP * UDP/IP * BSD socket interface	* UDP/IP * DDS * BSD socket interface
Execution Model/Server Management	* Every server process is created by Courier Daemon. (xnscourierd). * The daemon listens on behalf of all the servers for client's requests	* Light-weight process * Interrupt handler dispatch calls to server process based on dest, id in pkt. * Concurrent processes with shared memory and sync. primitives to service multiple clients * Excess idle server process created in peak time kill themselves	* Process approach * Static and pre-activated server process * dynamic server activated by inetd * Request from multiple clients are serialized and serviced in FIFO order	* Multi-thread (supported by a concurrent programming package)

Table 1 The Comparison of Courier, Cedar, Sun and Apollo RPC Implementations

	Mayflower RPC	*Athena RPC*	*Modula/V RPC*	*Rajdoot RPC*
OS/NOS/DOS	CDCS	BSD Unix (Athena)	Stanford V	BSD Unix
RPC Features	* Blocking	* Blocking * Non-Blocking	* Blocking	* Blocking
Semantics	* Maybe * At-most-once	* At-most-once	* At-most-once * At-least-once	* At-most-once
Orphan Treatment	NO	* NO (Instead Keep-alive message to detect whether the server is alive and reachable)	NO	* Deadlines * Crashcount * Terminator process
Security/ Authentication	NO	NO	NO	NO
Naming and Binding	* Dynamic * Using UID which is generated by compiler	* Name Server (servicename, inst -> hostaddr, port)	* Name Server (Module Identifier -> name and V processid, identity -> entry)	NO
Stub Generator and Interface	* NO (new syntax was added to the language)	* Interface service description and stub generator (cross-language RPC is possible)	* Remote module definition and stub generator	NO
Data Representation	* Superset of XDR & Courier	* No user defined operators and types		
RPC Exchange Protocol	* RR	* RR	* V IPC (sync. send and async receive)	
Transport Protocol/Interfa -ce	* UDP/IP (Ethernet) * Basic Block Protocol (Cambridge Ring)	* UDP * BSD socket interface	* VMTP	
Execution Model/Server Management		* Unix Process approach * Request from multiple clients are serialized and serviced in FIFO order * Instantiation of server process can be done manually, automatically using startup server or longlived	* Light-weight process approach * Dispatcher signal worker process * There is no good way for worker process to notify the dispatcher to reply or reply to caller on behalf of dispatcher process	* Manager process (well-known addr & stateless) listen and create process * For every newly created server, cross-check its clients crashcount value

Table 2 The Comparison of Mayflower, Athena, Modula/V, and Rajdoot RPC Implementations

4. Summary

Most of the RPC implementations are built on top of TCP/IP protocol suite, which use BSD socket IPC [14] [22] as a transport layer interface. Moreover, more than half of them actually implemented on Unix first, then probably ported to other operating systems. This is partly because of the maturity and popularity of Unix and TCP/IP. In future, the emphasis will be more on the performance improvement via various methods such as using light-weight process with share memory to avoid extra buffer copying between processes, specialized message-passing layer for RPC such as VMTP, or non-blocking RPC (with or without returned result). Both light-weight process and non-blocking RPC approaches help to achieve higher parallelism in both client and server. On the other hand, the reliability and fault-tolerant features of RPC will continue to be hot topics.

While the researchers are looking for improvement in RPC concepts and implementations, the industry is consolidating various RPC standards. There will be more effort in the development of distributed application programs based on RPC, once the Open Software Foundation (OSF) selects the components for its Distributed Computing Environments (DCE).

5. References

[1] Project Athena Executive Committee, "An Introduction to Project Athena", MIT, 1983.

[2] Almes, G., "The Impact of Language and System on Remote Procedure Call Design", *Proc. of 6th International Conference on Distributed Computing Systems*, May 1986, pp. 414-421.

[3] Geoff Arnold, "Internet Protocol Implementation Experiences in PC-NFS", *ACM Symp. on Oper. Syst. Principles*, Nov 1987, pp. 8-14.

[4] J.M. Bacon and K.G. Hamilton, "Distributed Computing with the RPC: the Cambridge Approach", *Distributed Processing*, IFIP, North-Holland, 1988, pp. 355-369.

[5] Bershad B.N., Ching D.T., Lazowska E.D., Sanislo J., Schwartz M., "A Remote Procedure Call Facility for Interconnecting Heterogeneous Computer Systems", *IEEE Trans. on Software Eng.*, Vol. SE-13, No. 8, Aug 1987, pp. 880-894.

[6] Birrell A.D. and Nelson B.J., "Implementing Remote Procedure Calls", *ACM Trans. on Computer Systems*, Vol. 2, No. 1, Feb 1984, pp. 39-59.

[7] Nelson, B.J., "Remote Procedure Call", *Tech. Rep. CSL-81-9*, Xerox Palo Alto Research Center, Palo Alto, Calif. 1981.

[8] Cheriton, D.R., "The V kernel: A Software Base for Distributed Systems", *IEEE Software*, Vol. 1, No. 2, 1984, pp. 19-42.

[9] Cheriton, D.R., "VMTP: A Transport Protocol for the next generation of Communication System", In *Sym. on Communication Architectures and Protocols*, ACM SIGCOMM, Aug 1986, pp. 406-415.

[10] Cooper, E.C., "Replicated Procedure Calls", *Proc. of 3rd ACM Symp. on Principles of Distributed Computing*, Aug 1984.

[11] Eric C. Cooper, "Circus: A Replicated Procedure Call Facility", *Proc. of 4th Symp. on Reliability in Distributed Software and Database Systems*, Oct 1984, pp. 11-24.

[12] Dineen, T. H., Leach, P.J., Mishkin, N.W., Pato, J.N., and Wyant, G.L., "The Network Computing Architecture and System: An Environment for Developing Distributed Applications", In *Proc. of the 1987 Summer USENIX Conference (Phoenix, Ariz., June)*, USENIX Association, Berkeley, Calif., 1987, pp. 385-398.

[13] Mike Kong, Terence H. Dineen, Paul J. Leach, Elizabeth A. Martin, Nathaniel W. Mishkin, Joseph N. Pato, and Geoffrey L. Wyant, *Network Computing System Reference Manual*, Prentice-Hall, Englewood Cliffs, N.J., 1990.

[14] Leffler, S.J., Fabry, R.S., W.N., Lapsley, P. Miller, S., and Torek, C., "An Advanced 4.3 BSD Interprocess Communication Tutorial", *Unix Programmer's Supplementary Documents*, Vol. 1 (PS1), 4.3 Berkeley Software Distribution, Computer Systems Research Group, Computer Science Division, Univ. of California, Berkeley, Calif., Apr 1986.

[15] Joshua Levy, "A Comparison of Commercial RPC Systems", Atherton Technology, 1989.

[16] Barbara Liskov, Dorothy Curtis, Paul Johnson, Robert Scheifler, "Implementation of Argus", *ACM Symp. on Oper. Syst. Principles*, 1987, pp. 111-122

[17] Barbara Liskov, "Distributed Programming in Argus", *Comm. of ACM*, Vol. 31, No. 3, Mar 1988, pp. 300-312.

[18] F. Panzieri and S.K. Shrivastava, "Rajdoot: A Remote Procedure Call Mechanism Supporting Orphan Detection and Killing", *IEEE Trans. on Software Eng.*, Vol. 14, No. 1, Jan 1988.

[19] Russel Sandberg, "The SUN Network File System: Design, Implementation and Experience", *Proc. of USENIX Conference*, 1986, pp. 150-166.

[20] M. Satyanarayanan and E.H. Siegel, "Multi-RPC: A Parallel Remote Procedure Call Mechanism", *Tech. Rep. CMU-CS-86-139/CMU-ITC-047*, Carnegie Mellon Univ. Aug 1986.

[21] M. Satyanarayanan and E.H. Siegel, "Parallel Communication in a Large Distributed Environment", *IEEE Trans. on Computers*, Vol. 39, No. 3, Mar 1990, pp. 323-348.

[22] Stuart Sechrest, "An Introductory 4.3BSD Interprocess Communication Tutorial", *Unix Programmer's Supplementary Documents*, Vol. 1 (PS1), 4.3 Berkeley Software Distribution, Computer Systems Research Group, Computer Science Division, Univ. of California, Berkeley, Calif., Apr 1986.

[23] Shrivastava S. and Panzieri F., "The Design of a Reliable Remote Procedure Call Mechanism", *IEEE Trans. Computers*, Vol. 31, No. 7, Jul 1982, pp.692-697.

[24] S.K. Shrivastava, "On the Treatment of Orphans in a Distributed System", *Proc. of 3rd Symp. Reliability Distributed Software and Database Systems*, Oct 1983, pp. 155-162.

[25] Alfred Z. Spector, "Perform Remote Operations Efficiently on a Local Computer Network", *Comm. of ACM*, Vol. 25, No. 4, Apr 1982, pp. 246-260.

[26] Robert J. Souza and Steven P. Miller, "Unix and Remote Procedure Calls: A Peaceful Coexistence?", *Proc. 6th IEEE International Conference on Distributed Computing Systems*, 1986.

[27] Sun Microsystems, "XDR: External Data Representation Standard (RFC 1014)", in *Internet Network Working Group Request for Comments*, No. 1014, Network Information Center, SRI International, Jun 1987.

[28] Sun Microsystems, "RPC: Remote Procedure Call Protocol Specification Version 2 (RFC 1057)", in *Internet Network Working Group Request for Comments*, No. 1057, Network Information Center, SRI International, Jun 1988.

[29] Sun Microsystems, "NFS: Network File System Protocol Specification (RFC 1094)", in *Internet Network Working Group Request for Comments*, No. 1094, Network Information Center, SRI International, Mar 1989.

[30] A. S. Tanenbaum and Robbert van Renesse, "Reliability Issues in Distributed Operating Systems", *Proc. of 6th Symp. on Reliability in Distributed Software and Database Systems*, Kingsmill Williamburg, VA, Mar 1987.

[31] Tanenbaum, A.S. and van Renesse, R., "A Critique of the Remote Procedure Call Paradigm", *Research into Networks and Distributed Applications* (Edt. by R. Speth), Elsevier Science Publishers B.V. (North-Holland), Apr 1988.

[32] Tanenbaum, A.S., Renesse, R. van, Staveren, H. van., Sharp, G.J., Mullender, S.J., Jansen, A.J., and Rossum, G. van, "Experiences with the Amoeba Distributed Operating System", *Report IR-194*, Dept of Mathematics and Computer Science, Vrije Universiteit, July 1989 (accepted for publication).

[33] Mullender, S.J., Rossum, G. van, Tanenbaum, A.S., Renesse, R. van, Staveren, H. van., "Amoeba - A Distributed Operating System for the 1990", To be published in *IEEE Computer*, May 1990.

[34] Steve Wilbur, Ben Bacarisse, "Building Distributed Systems with Remote Procedure Call", *Software Eng. Journal*, Sep 1987, pp. 148-159, also appeared in *UCL-CS TR 141*, Dept. of Comp. Sc., Univ. College London.

[35] Xerox Corporation, "Courier: The Remote Procedure Call Protocol", *Xerox System Integration Standard 038112*, Xerox OPD, Dec 1981.

[36] Yap, M., P. Jalote and S.K. Tripathi, "Fault Tolerant Remote Procedure Call", *Proc. of 8th International Conference on Distributed Computing Systems*, Jun 1988, pp. 48-54

[37] Lisa Zahn, Terence H. Dineen, Paul J. Leach, Elizabeth A. Martin, Nathaniel W. Mishkin, Joseph N. Pato, and Geoffrey L. Wyant, *Network Computing Architecture*, Prentice-Hall, Englewood Cliffs, N.J., 1990.