FERMILAB-Conf-89/135

Remote Procedure Execution Software for Distributed Systems *

Donald L. Petravick, Eileen F. Berman, and Gary P. Sergey
Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510 U.S.A.

May 1989

* Presented at ''Real Time Computer Applications in Nuclear, Particle, and Plasma Physics,'' Williamsburg, Virginia, May 16-19, 1989.

REMOTE PROCEDURE EXECUTION SOFTWARE FOR DISTRIBUTED SYSTEMS (*)

Donald L. Petravick, Eileen F. Berman, Gary P. Sergey
Online and Data Acquisition Software Groups
Fermi National Accelerator Laboratory
Batavia, Illinois 60510

Abstract

Remote Procedure Execution facilitates the construction of distributed software systems, spanning computers of various types. Programmers who use the RPX package specify subroutine calls which are to be executed on a remote computer. RPX is used to generate code for dummy routines which transmit input parameters and receive output parameters, as well as a main program which receives procedure call requests, calls the requested procedure, and returns the result. The package automatically performs datatype conversions and uses an appropriate connection oriented protocol.

Supported operating systems/processors are VMS(VAX), UNIX(MIPS R2000, R3000) and Software Components Group's pSOS (680x0). Connection oriented protocols are supported over Ethernet (TCP/IP) and RS232 (a package of our own design).

I. BACKGROUND

Experiment related computer systems at FERMILAB are evolving from single VAX processor systems to ones which use VAXstations in a Local Area VAX Cluster (LAVC). Front end FASTBUS and VME based 680x0 processors or MIPS/UNIX machines may also be part of this distributed system. Often individual VAXstations within the LAVC will perform separate tasks with the need to share or transfer data between each other and the VME or FASTBUS processor. Consistent and reliable data transfer between these systems involves knowledge of the individual communication call interfaces (DECnet, Ethernet, Terminal Lines) and the different data representations. Errors occurring within this multi-processor system would be hard to diagnose, as badly behaved non-interactive processes introduce non-obvious software failure points. It is essential that these processes detect error conditions and recover gracefully. Remote Procedure Execution is a high-level tool which provides all of these functions.

(*)sponsored by DOE Contract No.DE-AC02-76CH03000

II. WHAT IS REMOTE PROCEDURE EXECUTION

The goal when writing the Remote Procedure Execution (RPX) [4,5] software was to produce a reliable, easily portable package allowing transparent use of different CPU's and communications media for development of network application programs. RPX accomplishes this goal by using a layered approach. Layering enables RPX to insulate the user from the communication and data format routines. The user makes subroutine calls with no knowledge as to where the subroutine will actually be executed. No extraneous modifications to the user's source are required and the user programs and routines are unaware that they are intended for use across a network. To accomplish this separation RPX uses a client/server model to send requests and execute remote procedures.

While appropriate for new applications, RPX can be used to help migrate existing code to a new computer system. Isolating hard-to-port code behind procedure calls lets them continue to run under the old environment. For new applications RPX can be used as a prototyping tool. Since the software interface is specified as a procedure call, preliminary software can be prototyped and tested on a single computer. As the system grows the interface can be ported.

A. Current Applications

Currently at FERMILAB two applications are using the RPX software. The first application is a MIPS/UNIX based program which accesses a VMS based calibration database. The database can be initialized, opened, closed, read and written to from the UNIX machine across Ethernet. This application makes use of the greater computing power available on the MIPS machine for data manipulation.

The second application is a pSOS based message reporting package (MRPT). pSOS is a multi-tasking operating system kernel for 68000 family processor boards enhanced for FERMILAB's specific application needs. At FERMILAB, pSOS runs on a Motorola MVME133A (68020 CPU) VME processor board. The message reporting package is a subroutine library which is linked with pSOS application programs. Error messages generated by these programs are then passed to a VAX (via subroutine calls using RPX) where they can be displayed as desired.

B. Supported Configurations

The current release of the RPX software supports the following configurations -

- o VMS Client to VMS Server via TCP/IP across Ethernet
- o VMS Client to pSOS Server via TCP/IP across Ethernet
- o pSOS Client to VMS Server via TCP/IP across Ethernet
- o UNIX Client to VMS Server via TCP/IP across Ethernet

II. RPX INTERNALS

An assemblage of application independent routines constitute the layered foundation of RPX. All communication, error handling, and data transformation modules are functionally grouped and sufficiently independent to facilitate easy adaptability to a variety of communication media and protocols.

In an effort to simplify the RPX communication layer, I/O packages supporting connection oriented protocols are used. This design removes much of the overhead involved with network communications from RPX concerns, including flow control, assurance that data transferred is received, and notification of the demise of a partner.

Extensive measures were taken as part of the design phase, to identify all problematic situations that may arise during the execution of an RPX application. The resulting error handling layer not only encompasses detection, but utilizes a centralized reporting and recovery mechanism as well. For example, a lost connection or bad component thread would be reported and successfully recovered from so that subsequent requests could be handled without incident.

The transfer of data between various computers is complicated by the fact that binary representations of data may vary from machine to machine. For example, bytes of integer data may be ordered in memory from the least significant byte to the most significant byte or conversely from the most to the least significant byte.

RPX utilizes a set of macros to convert the internal representations of various datatypes into a standard transport form and also return the standard form back to the native machine representation. Datatypes supported include bytes, characters, 16 and 32 bit integers, and 32 bit floating point values. Arrays, pointers, and complex structures of the aforementioned are supported as well.

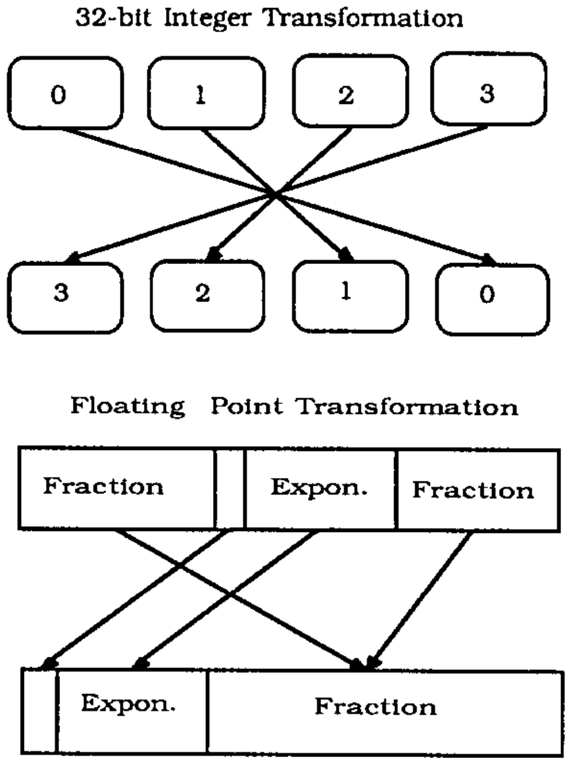

Fig 1. Transformations on Binary Dat

III. CLIENT/SERVER MODEL - USER INTERFACE

The underlying communication context for all RPX transactions may be viewed simply as a requesting process (or client) communicating with a receiving process (or server) at some remote network location. To attain this process-to-process relationship, client/server "stub" modules are created and linked with application code and the RPX run-time library. Tailored to the specifics of the designated remote procedures, the stubs efficiently bridge the gap between RPX internals and user written applications. Manipulation of parameter data to/from transport buffers, coordination of communications, and direction of data flow to appropriate destinations, are all managed by the stubs.

The client stub, which is linked with the user written mainline to form the client process, is comprised of routines which mimic the calling sequence of each subroutine that is to be called remotely. The server stub contains corresponding "shell" routines from which the actual application subroutines are invoked. The server process is created by linking the stub with the user's subroutine library.

IV. CLIENT/SERVER FLAVORS

The sequential nature of subroutine execution is simulated in the RPX environment by forcing the client process to remain idle until the server has completed the remote request and returned any parameter data. For most applications this scenario is acceptable, however the client/server model of RPX affords a degree of parallelism which was taken advantage of in the design and resulted in several options or flavors of procedure calls.

A. Concurrent Procedures

"Concurrent" procedures apply to remote subroutines which return no data. In this instance the client need not remain idle and hence is freed to execute simultaneously with the server.

B. Early Return Procedures

In certain situations it may be desirable to continue a remote procedure even after parameter data have been returned to the client. A user callable RPX routine is available to force an "early return" of parameters, allowing the client and server to run in parallel. The parameters are returned only once.

C. Deferred Procedures

Useful in applications such as resource managers, "Deferred" procedures provide an opportunity to set aside a particular call until further notice. The procedures may be resumed under program control or after timeout of a specified interval. This is decidedly non-transparent, but still within the RPX framework.

D. Client/Server Configurations

The multiple client/server configurations highlight the distributed system capabilities available with RPX. Servers readily field calls from both single or multiple clients distributed across a network. So too, may clients invoke procedures residing on separate network server destinations. The configurations are dictated by the specific needs of the intended application and are limited only by the constraints of the network involved.

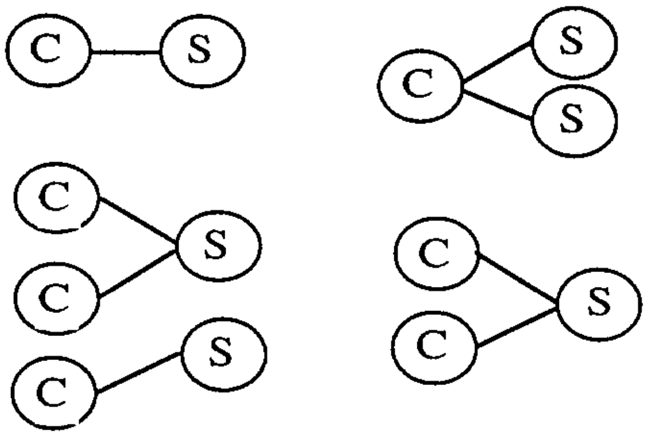

Fig. 2. Potential Client - Server Configurations

V. CONCLUSION

RPX provides a consistent, transparent, easy to use method for executing procedures located on a remote node. Communication and data representation problems are completely handled by RPX removing them from the user's concern. Due to its layered approach, RPX is easily ported to new operating systems. Future plans include implementation of the following -

- o Support for connection oriented terminal line communications.
- o Support for DECnet communications.
- o Support for the GPM (a Struck General Purpose FASTBUS Master with Motorola 680x0 processor).
- o Support for MIPS/UNIX servers.
- o Automated stub creation.

VI. ACKNOWLEDGMENTS

During research at the onset of the RPX project, extensive evaluations of the CERN Remote Procedure Call (RPC) package and the SUN Microsystems RPC package were conducted [1,2,3]. We are especially grateful to Tim Berners-Lee for his help with examining CERN's RPC package and for the ideas behind the Client/Server stubs, concurrent procedure calls and early return.

VII. REFERENCES

[1] T. Berners-Lee, "RPC User Manual", CERN DD/OC

[2] T.Berners-Lee, "RPC System: Implementation Guide", CERN DD/OC.

[3] Sun Microsystems (Trademark), "Remote Procedure Call Programming Guide".

[4] E. Berman, D. Petravick, G. Sergey, "Design Outline For RPX", Fermilab Computing Department Note DS-166.

[5] E. Berman, D. Petravick, G. Sergey, "RPX User's Guide", Fermilab Computing Department Note PN-384.