Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

APM Business Unit

Using Remote Procedure Call Standards

Chris Mayers

Abstract

Organizations defining their own architectures wish to determine which Remote Procedure Call (RPC) standards are applicable for use.

This briefing note, originally prepared for a client project is a review of RPC standards within the X/Open Architectural Framework, and is a contribution to the Network Services category of Application Platform Services.

It specifically explores the relationship of RPC standards to other categories of Application Platform Services.

APM.1653.00.02 **Draft** 24th March 1996

Briefing Note

Distribution:
Supersedes:
Superseded by:

Using Remote Procedure Call Standards

Using Remote Procedure Call Standards

Chris Mayers

APM.1653.00.02

24th March 1996

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(01223) 515010
INTERNATIONAL	+44 1223 515010
FAX	+44 1223 359779
E-MAIL	apm@ansa.co.uk

Contents

1 Introduction

1.1 Audience

This document is intended for those selecting or procuring products that make use of Remote Procedure Call (RPC). It is particularly aimed at those applying the X/Open Architectural Framework [X/Open].

1.2 Scope

This document covers current standards for RPC. It concentrates on the functional aspects of these standards.

This document discusses non-functional aspects of these standards (for example, performance and security), but recommendations on these non-functional aspects are outside the scope of this document.

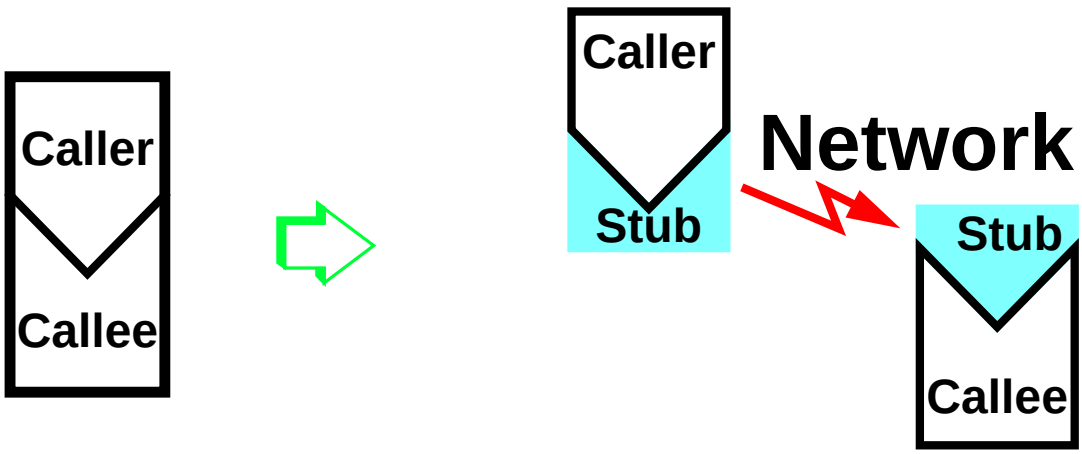
1.3 Status

This version is a draft document, for review.

2 Overview of Remote Procedure Call Standards

2.1 Remote Procedure Call as a general technique

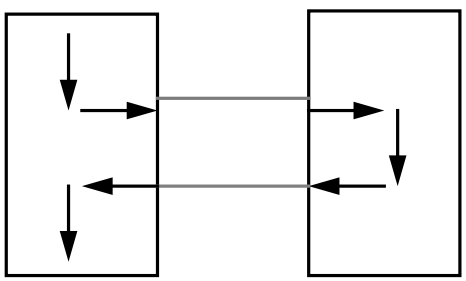
Remote Procedure Call (RPC) is not a product or a single standard - it is a general technique for interprocess communication.



In an ordinary procedure call in a conventional programming language (such as a C function or FORTRAN subroutine), a program invokes a procedure, supplying a set of parameters as arguments, and receiving one or results. The called procedure is part of the same program as the called procedure.

In a remote procedure call (RPC), the calling program and the called procedure can reside on different machines. Special logic, known as stubs, is included in both the calling program and the called program.

In the calling program, a stub marshalls the arguments and transmits them across the network to the remote machine. The corresponding stub at the remote then unmarshalls the arguments, and invokes the called procedure. When the called procedure completes, its results are transmitted back to the calling program via the stubs in the same way.



The calling program is often referred to as the client, and the called program as the server, so RPC programming is one style of client/server programming.

RPC is an attractive technique, because to the application programmer it is similar to an ordinary procedure call. Application programmers do not have use communications services to transmit and receive data; this is handled automatically by the stubs.

Applications programmers do not need to learn a new programming language to use RPC. Indeed, the client and server portion can be written in different programming languages if this is appropriate. This is similar to the idea of calling a library of FORTRAN subroutines from a C program using an ordinary procedure call.

And, just as a subroutine library can be used by many programs, the server program can be used by many different client applications.

The RPC technique can be used to wrap up a large and complex legacy application, to provide its service to any other application that needs it, anywhere in the network.

For more information on RPC generally, refer to [RPC].

Note: This book is an excellent primer, but the standards information in it is now rather out-of-date.

2.2 RPC within the X/Open Architectural Framework

2.2.1 RPC within Network Services

RPC logically falls within the Network Services Application Platform Services category of the X/Open Technical Reference Model. It underpins the Distributed Computing services.

The following Network Services specifications directly relate to RPC [X/Open]:

- G212: Distributed Computing Services (XDCS) Framework
- C309: X/Open DCE: Remote Procedure Call

2.2.2 Relationship with other Application Platform Services

Since RPC is a generally applicable technique, specifications for other current and planned Application Platform Services already use different RPC standards. For example:

- Object Services: the Object Request Broker depends on RPC mechanisms
- Transaction Processing: T008 Transaction Processing uses the TxRPC transactional remote procedure call interface

2.2.3 Relationship with other Network Services

Note: Explain positioning (session layer) here

RPC can potentially be used over any communications transport protocol (for example the Internet or OSI transport protocols); products differ in which they support.

However, most RPC implementations assume that a LAN connection is being used. Client/server operation is not universally possible, unless systems are designed specifically to operate under limited bandwidth constraints.

Such constraints will be reflected in the quality-of-service (QoS) offered by the connection. RPC is still a valid approach for this situation, provided that

specially-modified RPC implementations and communications transport protocols support:

- quality-of-service negotiation
- one-way communications (for example, satellite broadcasting or paging)
- low-bandwidth or intermittent communications (for example, dial-up or wireless)

Quality-of-service negotiation is not supported in open standard communications transports protocols today. However, new transport protocols are the subject of much active research, and COTS results can be expected with the next few years.

This is a study topic under the ANSA programme.

2.3 RPC applicability

RPC is a powerful technique, appropriate for communications between applications that requires short response times and relatively small amounts of data transfer.

RPC is not appropriate for:

- continuous information flows or streams (for example, speech, video, or telemetry)
- off-line communication where the receiver is not always available
- bulk data transfer

Although RPC could in principle be used for bulk data transfer (for example image file transfer), many implementations limit the size of data in one RPC, typically to a few Kbytes. So other techniques are generally used instead.

It should be noted that the use of encapsulated objects (as recommended by the ANSA architecture) will tend to reduce the amount of data transfer in the future.

2.4 RPC interoperability and portability

2.4.1 RPC interoperability

Note: To be determined

Note: Also need to consider impact of IPv6 (see RFC1752).

2.4.2 RPC portability

Note: Need to consider IDL+Programming Language mapping as a unit for source-code portability

Note: Some RPCs have a direct API (~DII), as well as an IDL

Note: Lower-level API usage (transport APIs) also affects portability as a whole

2.5 RPC non-functional characteristics

Note: Need to consider at least performance and security here

It is questionable whether security requirements should influence RPC standards directly, or whether this should be an issue for higher-level services, independent of RPC standards.

2.6 RPC and transparency

RPC is similar to an ordinary procedure call as far as the application programmer is concerned, but not identical.

There are differences, and it is important that the applications programmer is aware of them. Some of these differences will have a direct impact on the application programmer, some will not.

Some differences can be hidden from the application programmer; this aspect is then said to be transparent to the application programmer. The underlying mechanisms that achieve this are transparency mechanisms, or transparencies.

To understand this, it is helpful to discuss some of the differences in more detail.

2.6.0.1 Diversity(heterogeneity) and access transparency

The client and server programs generally execute on different machines. These machines may internally represent data in different formats. This is a form of diversity, also known as heterogeneity.

For example, different manufacturers' machines store the bytes that represent a (32-bit) integer in different orders:

Table 2.1: Byte ordering for integers

CPU	Ordering
Intel 80x86	b0 b1 b2 b3
Digital PDP-11	b2 b3 b0 b1
Digital VAX-11	b0 b1 b2 b3
Motorola M68K	b3 b2 b1 b0

Floating-point formats, data packing, and data alignment rules also differ between machines.

For the client and server to communicate, they must use compatible data formats. It is the responsibility of the client and server stubs to convert the internal data formats into a standard data format before transmission over the network. One such standard data format is Sun's XDR (eXternal Data Representation).

In a layered communications protocol such as the OSI protocol, conversion into a standard data format is a function of the presentation layer. In the OSI protocol, these standard data formats are defined in Abstract Syntax Notation 1 (ASN.1), usually using the Basic Encoding Rules (BER).

However, most RPC protocols are not layered on the OSI communications protocols, and so do not use ASN.1. Instead, the client and server stubs carry out the data format conversion directly. Some RPC systems allow a choice.

In either case, the application programmer need not be aware of the different data formats. The client and server stubs provide a transparency; access

transparency. Access transparency also masks the communications mechanisms and communications protocols from the application programmer.

2.6.0.2 *Remoteness and location transparency*

As client and server programs execute on different machines, they are said to be remote from each other. This has two implications; one obvious, one less so.

The obvious implication is that RPC will be slower than an ordinary procedure, because of the communications delay in transmitting the arguments to the server, and transmitting the results back again. RPC performance may or may not be important to the application. If RPC performance is important, specialist advice should be sought as early as possible during system development, particularly for systems involving wide-area communications.

The less obvious implication of remoteness is that the client program needs to know on which machine the server program is running. This information could be supplied as an extra parameter to the RPC, by the application programmer. This would have has two disadvantages.

First, the application programmer needs to construct a network address for the machine the server program is running on. The format of this address will depend on the communications protocol being used. A typical Internet address would be “192.5.254.48”; a typical SMB address would be “MYSERVER”. If the application programmer has to handle network address formats, this would breach access transparency.

Second, the application programmer needs to know the network address itself. How is the information supplied? It is unacceptable for application programmer to code this information directly into the client program. If the server program is moved to a different machine, the client program will stop working. Or, suppose that the client program is to be used at several sites with separate servers, each with a different address.

The conclusion is that the client application should not need to know the location of the server. Instead, a location transparency mechanism should provide this information to the client stub on the application’s behalf; the client application can obtain location information from a name server (a particular kind of directory service).

There are also extensions of location transparency; relocation transparency masks the fact that a server program has moved since it was last used, and migration transparency masks the fact the server program is being moved whilst still in use. These extensions are not normally provided by RPC systems; relocation transparency is desirable for many applications.

To support relocation transparency, directory services are needed. No such services are identified within the X/Open Architectural Framework.

2.6.0.3 *Other transparency requirements*

Access transparency and location transparency should be regarded as the minimum requirements for RPC network services. All RPC products provide access transparency; some provide location transparency.

However, there are more potential requirements for transparency; not only relocation and migration transparency, but also:

- failure transparency, which masks the failure of the server, using fault tolerance techniques

- replication transparency, which masks the fact that a group of servers are acting together to provide the service, to share the load, or for fault tolerance
- transaction transparency, which masks the fact that other client applications may be using the same server at the same time, updating the same data

This list is not exhaustive. But even these cannot be met by an RPC network service alone. For instance, failure transparency needs the support of the server program, to recover from failures.

Therefore, network services are needed that use RPC as a basic technique, but go beyond it to provide the more sophisticated transparencies. OSF/DCE, CORBA, and ANSAware distributed programming environments aim to do just this. It is at this stage that distributed object technology becomes important. This is discussed in the next chapter.

2.7 History of RPC standards and COTS products

Since the mid 1980s, proprietary RPC facilities have been available on many platforms, notably on Unix. Two of the most popular were Sun's ONC RPC and HP/Apollo's NCS RPC. With the rise of the IBM PC in mid to late 1980s, these RPC facilities were also made available there. ONC RPC and NCS RPC are not compatible with each other. Almost all platforms now support some kind of RPC; this may be bundled with the operating system, or supplied by a third party.

2.7.1 RPC in OSF DCE

The RPC used in OSF/DCE was based on HP/Apollo's NCS, but also used elements of Sun's ONC. It is not fully compatible with either. The RPC protocol is fully defined by the DCE specification. The DCE RPC has also been adopted by X/Open.

2.7.2 RPC in OMG CORBA

The RPC protocol to be used in CORBA was much debated. The OMG eventually decided to support an RPC framework known as Universal Networked Objects (UNO). This framework allows for multiple RPC protocols. Proprietary RPC protocols are permitted within CORBA, but CORBA products must also support a specific UNO protocol (based on TCP/IP) to allow interworking between different CORBA products. Another UNO protocol, DCE-CIOP, using DCE RPC, has also been specified; this will permit RPC interworking with DCE systems.

2.7.3 Other RPC international standards

ECMA-127 ("Basic Remote Procedure Call Using OSI Remote Operations") was standardized in December 1987. It is not known whether this standard has been used widely.

2.7.4 Other RPC products

Other proprietary RPC protocols also exist. These do not satisfy the criteria for X/Open standards and specifications; they are not recommended (except for circumstances where there is no alternative).

3 Table of Recommended Standards

3.1 Network Protocols - ONC (Open Network Computing) RPC and XDR

3.1.1 Specification Title

RFC 1057 - Remote Procedure Call Protocol Specification Version 2
RFC 1014 - XDR: External Data Representation Standard

Note: RFC 1800 lists RFC 1057 as an Informational Protocol. (“Informational Protocols have no status”.) RFC 1800 does not list RFC1014 at all. RFC 1800 lists RFC 1050 (RPC Version 1) as a Historic Protocol. (“Historic protocols have Not Recommended status.”)

3.1.2 Applicability

ONC RPC provides a general-purpose Remote Procedure Call mechanism over connection-oriented (TCP/IP) or connectionless (UDP/IP) transport protocols. It can therefore be used for any application where these protocol provide acceptable latency (response time).

3.1.3 Stability

The ONC specification is mature. However the above specifications are being replaced by:

RFC 1831 - Remote Procedure Call Protocol Specification Version 2
RFC 1832 - XDR: External Data Representation Standard

These new specifications are upwards-compatible revisions of the correspond standards above. Note that the Protocol Specification is still Version 2.

By comparing the standards, it appears that the revisions are:

- some new error codes (RFC 1831)
- more generic support for authentication mechanisms (RFC 1831)

Note: These authentication mechanisms are the subject of an Internet Draft. It remains to be seen whether they progress to an RFC.

- signed and unsigned hyper (64-bit) integer (RFC 1832)
- quadruple-precision(128-bit) floating-point (RFC 1832)

An additional specification is:

RFC 1833 - Binding Protocols for RPC Version 2

This specification was previously an appendix to RFC 1057.

A newer set of ONC protocols, ONC+ has been developed by Sun Microsystems Inc. These protocols are backwards-compatible with the ONC RPC. The main advantage of ONC+ is that it is transport-independent (unlike ONC). ONC+ forms part of Sun’s Solaris operating system; ONC+ is licensed by Sun to other vendors. Since ONC+ offers no further benefits over ONC, it is recommended

that ONC+ products should be procured and used as ONC-compatible products.

Note: It is unclear whether Sun's ONC+ conforms to RFC 1831/1832/1833, but a brief inspection of the Sun RPC programs supplied with Solaris 2.4 indicates that it probably does.

3.1.4 Level of Consensus

ONC is a de-facto standard on Unix platforms.

ONC has not yet been adopted as an international standard in its own right.In fact RFC1790 is an official public record of an agreement between the Internet Society and Sun Microsystems Inc. to permit the flow of the above two specifications into the Internet standards process.

Note: This might explain the appearance of RFC 1831 and 1832; see above.

3.1.5 Product Availability

Products are widely available for all platforms.

On some platforms, ONC is bundled with the operating system. (This is typical for Unix, since the Unix network filing system NFS uses ONC RPC._

3.1.6 Benefits

1. A mature protocol, with proven interoperability
2. Supports TCP/IP and UDP/IP transport protocols, hence can be used over medium-speed wide-area networks.

3.1.7 Risks

1. ONC has no provision for quality-of-service
2. ONC is not support as an RPC standard by some other X/Open services (Object Services RPC or Distributed Computing RPC).
3. ONC may not operate efficiently over very-high-speed or very-low-speed connections
4. ONC supports only TCP/IP and UDP/IP transport protocols
5. ONC has limited support for security

3.1.8 Recommendations

ONC is an appropriate X/Open RPC standard for use when all the following apply:

1. No Object Services are required
2. No Transaction Processing services are required
3. Security services are not required (or are provided by means outside the RPC protocol
4. Either of the NFS Network Services are already in use:
   - — “D030 Protocols for X/Open PC Interworking: (PC) NFS”
   - — “C218 Protocols for X/Open Interworking XNFS Issue 4”

3.2 Network Protocols - X/Open DCE RPC

3.2.1 Specification Title

C309 X/Open DCE: Remote Procedure Call

3.2.2 Applicability

DCE RPC provides a conventional Remote Procedure Call mechanism as part of the Distributed Computing Environment (DCE). The Remote Procedure Call mechanism supports the Kerberos authentication and authorization mechanisms.

The DCE RPC can be used anywhere the DCE as a whole can be used.

3.2.3 Stability

Note: To be determined

3.2.4 Level of Consensus

International standard, already adopted by X/Open as part of the Distributed Computing Services (XDCS) Framework.

The Object Management Group (OMG) have adopted the DCE RPC in their Interoperability framework as an optional Environment-Specific Inter-ORB Protocol (ESIOP).

DCE RPC has been adopted by Microsoft as part of their Component Object Model for distributed computing.

3.2.5 Product Availability

Products are available from many operating system vendors.

3.2.6 Benefits

1. Underpins the OSF's Distributed Computing Environment (DCE), being adopted by X/Open.
2. Can be used with Object Services (CORBA DCE-ESIOP).
3. Has been adopted by Microsoft to underpin Distributed OLE.
4. Incorporates Kerberos security.

3.2.7 Risks

1. Not yet proven over wide area networks.
2. Client-side and server-side implementations can require much memory.
3. Application Programming Interface (API) is complex for programmers.
4. Not all products are yet mature.
5. Some vendors do not sell the DCE RPC alone, but only as part of the complete DCE.

3.2.8 Recommendations

DCE RPC is an appropriate RPC standard if:

1. The X/Open DCE is being used, or there is a need to interoperate with it

2. The support for security is considered acceptable

3.3 Network Protocols - CORBA IIOP

3.3.1 Specification Title

CORBA Internet Inter-ORB Protocol (IIOP)

3.3.2 Applicability

The CORBA Internet Inter-ORB Protocol (IIOP) is intended for interoperability between different CORBA Object Request Broker (ORB) implementations.

The CORBA IIOP is a specific mapping of a CORBA General Inter-ORB Protocol (GIOP) onto the TCP/IP protocol.

Unlike the other RPCs, the CORBA IIOP does not provide for security as part of the RPC itself. Instead this is provided by a separate Security Object Service.

3.3.3 Stability

Note: To be determined. It is understood that the CORBA 2/Interoperable specification, of which IIOP is a part, has a number of ambiguities in the specification that require resolving.

3.3.4 Level of Consensus

The CORBA IIOP has been adopted by the Object Management Group (OMG), and is expected to progress to an international standard.

The CORBA IIOP forms part of the CORBA/Interoperable conformance testing point.

3.3.5 Product Availability

Products are becoming available with demonstrated interoperability.

3.3.6 Benefits

1. Interoperability between CORBA ORB implementations.
2. A lightweight RPC protocol, permitting efficient implementations.

3.3.7 Risks

1. Implementations are not yet mature.
2. Requires the TCP/IP protocol; no other transport protocols are yet supported. (Mappings of the General Inter-ORB Protocol to other connection-oriented transport protocols are under study by the OMG.)
3. Not yet proven over wide-area networks.

3.3.8 Recommendations

The CORBA IIOP is an appropriate RPC standard if:

1. Interoperability is required between CORBA ORBs

2. The TCP/IP protocol is acceptable
3. Security is provided by means outside the RPC protocol.

References

[ISO/IEC 10746]

Basic Reference Model for Open Distributed Processing Parts 1-4

[OMG 92]

Object Management Architecture Guide, Object Management Group, Inc., 2nd. ed (1992)

[RPC]

Power Programming with RPC, John Bloomer, O'Reilly & Associates Inc. ISBN 0-937175-77-3

[X/Open]

X/Open Architectural Framework
Draft 1.2