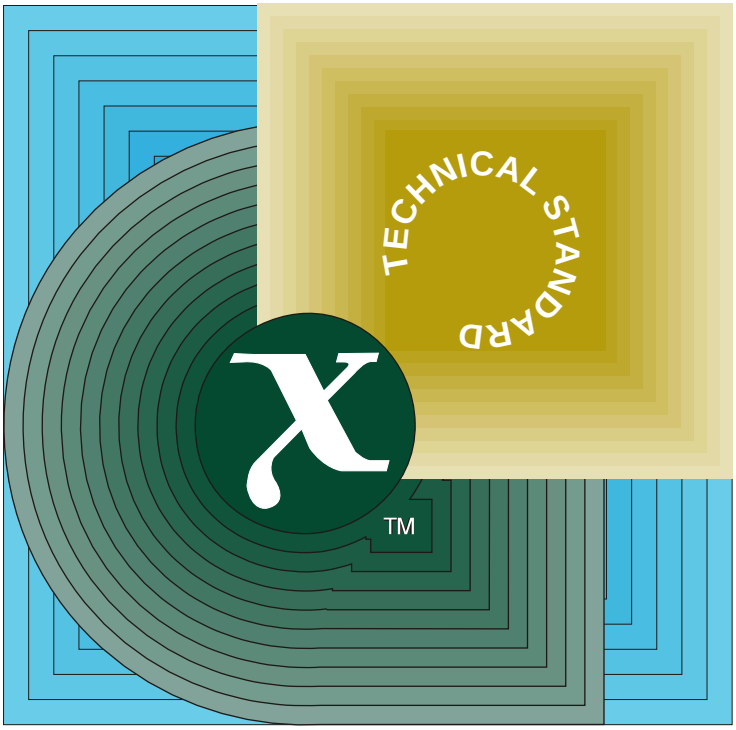
Technical Standard

Distributed Transaction Processing: The TxRPC Specification



THE Open GROUP

[This page intentionally left blank]

X/Open CAE Specification

Distributed Transaction Processing:

The TxRPC Specification

X/Open Company Ltd.

X/Open CAE Specification

Distributed Transaction Processing: The TxRPC Specification

ISBN: 1-85912-115-2
X/Open Document Number: C505

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.org

Contents

List of Figures

List of Tables

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

- a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.

- a new *Issue* does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains *both* the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done either by email to the X/Open info-server or by checking the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information by email, send a message to info-server@xopen.co.uk with the following in the Subject line:

```
request corrigenda; topic index
```

This will return the index of publications for which Corrigenda exist.

This Document

This document is a CAE specification (see above). It defines the TxRPC interface which is the application programming interface to a communication resource manager. It also defines the Remote Task Invocation (RTI) Service Definition and Protocol Specification which together form the Application Service Element (ASE) for TxRPC.

This specification is designed to be used in conjunction with the X/Open DCE: Remote Procedure Call specification (see the referenced X/Open **DCE RPC** specification). Readers are expected to have a thorough understanding of the following sections of that specification:

- Part 1: Remote Procedure Call Introduction
  - — Chapter 1, Introduction to the RPC Specification
- Part 3: Interface Definition Language and Stubs
  - — Chapter 4, Interface Definition Language
  - — Chapter 5, Stubs (ignore pipes)
- Part 4: RPC Services and Protocols
  - — Chapter 6, Remote Procedure Call Model
  - — Chapter 7, RPC Service Definition
  - — Chapter 8, Statechart Specification Language Semantics
  - — Chapter 9, RPC Protocol Definitions
  - — Chapter 11, Connection-oriented RPC Protocol Machines
  - — Chapter 12, RPC PDU Encodings (ignore Connectionless sections)
  - — Chapter 14, Transfer Syntax NDR
  - — Appendix A, Universal Unique Identifier
  - — Appendix E, Reject Status Codes and Parameters
  - — Appendix F, IDL to C-language Mappings

— Appendix I, Protocol Identifiers

— Appendix K, Architected and Default Values for Protocol Machines

— Appendix L, Protocol Tower Encoding

— Appendix N, IDL Data Type Declarations

— Appendix P, Conversation Manager Interface Definition.

In addition, those planning to implement an RPC TxRPC CRM (as opposed to an IDL-only TxRPC CRM), need to be familiar with:

- Part 2: RPC Application Programmer's Interface
  - — Chapter 2, Introduction to the RPC API
  - — Chapter 3, RPC API Manual Pages.

The structure of the **TxRPC** specification (this document) is as follows:

- Part 1: TxRPC Communication Application Programming Interface (API)
  - — Chapter 1 is an introduction to both the TxRPC API and the ASE.
  - — Chapter 2 provides an introduction to the X/Open DTP model and fundamental definitions for the API.
  - — Chapter 3 is an overview of the TxRPC API.
  - — Chapter 4 summarises the implications of the TxRPC API on implementors.
- Part 2: TxRPC Application Service Element (ASE)
  - — Chapter 5 is an introduction to the RTI model.
  - — Chapter 6 defines the RTI Application Context.
  - — Chapter 7 defines the RTI Service Definition.
  - — Chapter 8 specifies the RTI Protocol Machine (RTI-PM).
  - — Chapter 9 contains known architectural constants that relate to RTI.
- Part 3: API Appendices
  - — Appendix A contains an example of the use of the RPC TxRPC interface.
  - — Appendix B contains an example of the use of the IDL-only TxRPC interface.
  - — Appendix C describes the mapping between the TxRPC API and RTI.
- Part 4: ASE Appendices
  - — Appendix D contains the mapping from the terminology used in the description of the RPC-ASE to the terminology of OSF RPC.
  - — Appendix E provides examples of the usage of RTI.

There is a glossary and an index at the end.

Intended Audience

Parts 1 and 3 of this document are intended for application programmers who wish to write portable programs that use global transactions and that communicate using the Remote Procedure Call paradigm. The whole document is of interest to implementors of the TxRPC application programming interface.

All readers are expected to be familiar with the X/Open documents **Distributed Transaction Processing: Reference Model** and **Distributed Transaction Processing: The TX (Transaction Demarcation) Specification**. Implementors are also expected to be familiar with the X/Open document **Distributed Transaction Processing: The XA Specification** and the ISO Open Systems Interconnection (OSI) standards listed in **Referenced Documents** on page xvii.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for filenames, keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — variable names, for example, substitutable argument prototypes and environment variables
  - — C-language functions; these are shown as follows: *name*( )
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a C-language header file.
- Names surrounded by braces, for example, {ARG_MAX}, represent symbolic limits or configuration values, which may be declared in appropriate C-language header files by means of the C **#define** construct.
- The notation [ABCD] is used to identify a coded return value in C.
- Syntax and code examples are shown in `fixed width` font.
- Variables within syntax statements are shown in *`italic fixed width`* font.

Where there are explicit references to terms used in the OSI standards, they are shown as in the OSI standard; this means that OSI parameter values are shown in double quotes even when they are not string literals.

Trade Marks

X/Open® is a registered trade mark, and the ‘‘X’’ device is a trade mark, of X/Open Company Limited.

Referenced Documents

The following standards are referenced in this specification:

ASN.1
: ISO 8824: 1990 Information Technology — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1).

BER
: ISO / IEC 8825: 1990 (ITU-T Recommendation X.209 (1988)), Information Technology — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).

ISO C
: ISO / IEC 9899: 1990, Programming Languages — C (technically identical to ANSI standard X3.159-1989).

ISO 8649
: ISO 8649: 1988, Information Processing Systems — Open Systems Interconnection — Service Definition for the Association Control Service Element.

ISO 8650
: ISO 8650: 1988 Information Processing Systems — Open Systems Interconnection — Protocol Specification for the Association Control Service Element.

ISO / IEC 9545
: ISO / IEC 9545: 1989, Information Technology — Open Systems Interconnection — Application Layer Structure.

ISO / IEC 9804
: ISO / IEC 9804: 1994, Information Technology — Open Systems Interconnection — Service Definition for the Commitment, Concurrency, and Recovery Service Element.

ISO / IEC 9805
: ISO / IEC 9805: 1994, Information Technology — Open Systems Interconnection — Protocol Specification for the Commitment, Concurrency, and Recovery Service Element.

OSI TP
: ISO / IEC 10026, Information Technology — Open Systems Interconnection — Distributed Transaction Processing, Parts 1 to 5:

  Part 1: 1992, OSI TP Model
  Part 2: 1992, OSI TP Service
  Part 3: 1992, Protocol Specification
  Part 4: 1995, Protocol Implementation Conformance Statement (PICS) proforma
  Part 5: DIS 1993, Application context proforma and guidelines when using OSI TP.

OSI TP Profiles
: ISO / IEC ISP 12061: 1995, Information Technology — Open Systems Interconnection — International Standardized Profiles: OSI Distributed Transaction Processing, Parts 6 and 8:

  Part 6: Application supported transactions — Shared control (ATP12)
  Part 8: Provider supported unchained transactions — Shared control (ATP22).

The following X/Open documents are referenced in this specification:

CPI-C, Version 2
: X/Open CAE Specification, November 1995, Distributed Transaction Processing: The CPI-C Specification, Version 2 (ISBN: 1-85912-135-7, C419).

DCE RPC
: X/Open CAE Specification, August 1994, X/Open DCE: Remote Procedure Call (ISBN: 1-85912-041-5, C309).

DTP
: X/Open Guide, November 1993, Distributed Transaction Processing: Reference Model, Version 2 (ISBN: 1-85912-019-9, G307).

TX
: X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN: 1-85912-094-6, C504).

XA
: X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN: 1-872630-24-3, C193 or XO/CAE/91/300).

XA+
: X/Open Snapshot, July 1994, Distributed Transaction Processing: The XA+ Specification, Version 2 (ISBN: 1-85912-046-6, S423).

XAP-TP
: X/Open CAE Specification, April 1995, ACSE/Presentation: Transaction Processing API (XAP-TP) (ISBN: 1-85912-091-1, C409).

XATMI
: X/Open CAE Specification, October 1995, Distributed Transaction Processing: The XATMI Specification (ISBN: 1-85912-130-6, C506).

XDCS
: X/Open Guide, November 1992, Distributed Computing Services (XDCS) Framework (ISBN: 1-872630-64-2, G212).

X/Open CAE Specification

Part 1:
TxRPC Communication Application Programming Interface (API)

X/Open Company Ltd.

Chapter 1

Introduction

This chapter provides an outline of the X/Open Distributed Transaction Processing Model and explains the position of this specification as one of the Communication Resource Manager (CRM) interfaces.

1.1 X/Open DTP Model

The X/Open Distributed Transaction Processing (DTP) model is a software architecture that allows multiple application programs to share resources provided by multiple resource managers, and allows their work to be coordinated into global transactions.

The X/Open DTP model comprises five basic functional components:

- an Application Program (AP), which defines transaction boundaries and specifies actions that constitute a transaction
- Resource Managers (RMs) such as databases or file access systems, which provide access to resources
- a Transaction Manager (TM), which assigns identifiers to transactions, monitors their progress, and takes responsibility for transaction completion and for coordinating failure recovery
- Communication Resource Managers (CRMs), which control communication between distributed applications within or across TM domains
- a communication protocol, which provides the underlying communication services used by distributed applications and supported by CRMs.

X/Open DTP publications based on this model specify portable Application Programming Interfaces (APIs) and system-level interfaces that facilitate:

- portability of application program source code to any X/Open environment that offers those APIs
- interchangeability of TMs, RMs and CRMs from various sources
- interoperability of diverse TMs, RMs and CRMs in the same global transaction.

Chapter 2 defines each component in more detail and illustrates the flow of control.

1.2 X/Open Communication Resource Manager Interfaces

An important aspect of distributed transaction processing applications is communication. Within the product domain for DTP tools, there are several popular communication paradigms in common use today or expected to be in common use in the future. The communication paradigm chosen can significantly influence the architecture of the application. The unique strengths of each paradigm make it attractive for specific applications.

The referenced **DTP** guide defines a functional component known as a Communication Resource Manager (CRM), which provides access to a communication medium between applications.

Because it is not possible to choose a single communication paradigm applicable to the entire broad range of DTP applications, X/Open provides application programming interfaces (APIs) for the most popular paradigms in order to bring the benefits of open systems to the widest possible range of transaction processing applications. These are the request/response paradigm and the conversational paradigm.

Many applications already running on open systems use the request/response paradigm. X/Open specifications for this paradigm are the IDL-based TxRPC CRM interface (see this document) and the library-based XATMI CRM interface (see the referenced **XATMI** specification). TxRPC fits within the context of the X/Open Distributed Computing Services Framework (XDCS) and allows application writers to invoke remote procedure calls (RPCs) in the same form as local procedures, but with transaction semantics.

For applications choosing to use the conversational paradigm, where communication takes place through an application-defined exchange of messages, X/Open offers the library-based interfaces XATMI (see the referenced **XATMI** specification) and CPI-C (see the referenced **CPI-C** specification).

Chapter 2

Model and Definitions

This chapter discusses the TxRPC interface in general terms and provides necessary background material for the rest of the specification. The chapter shows the relationship of the interface to the X/Open DTP model. The chapter also states the design assumptions that the interface uses and shows how the interface addresses common DTP concepts.

2.1 X/Open DTP Model

The boxes in the figure below are the functional components and the connecting lines are the interfaces between them. The arrows indicate the directions in which control may flow.

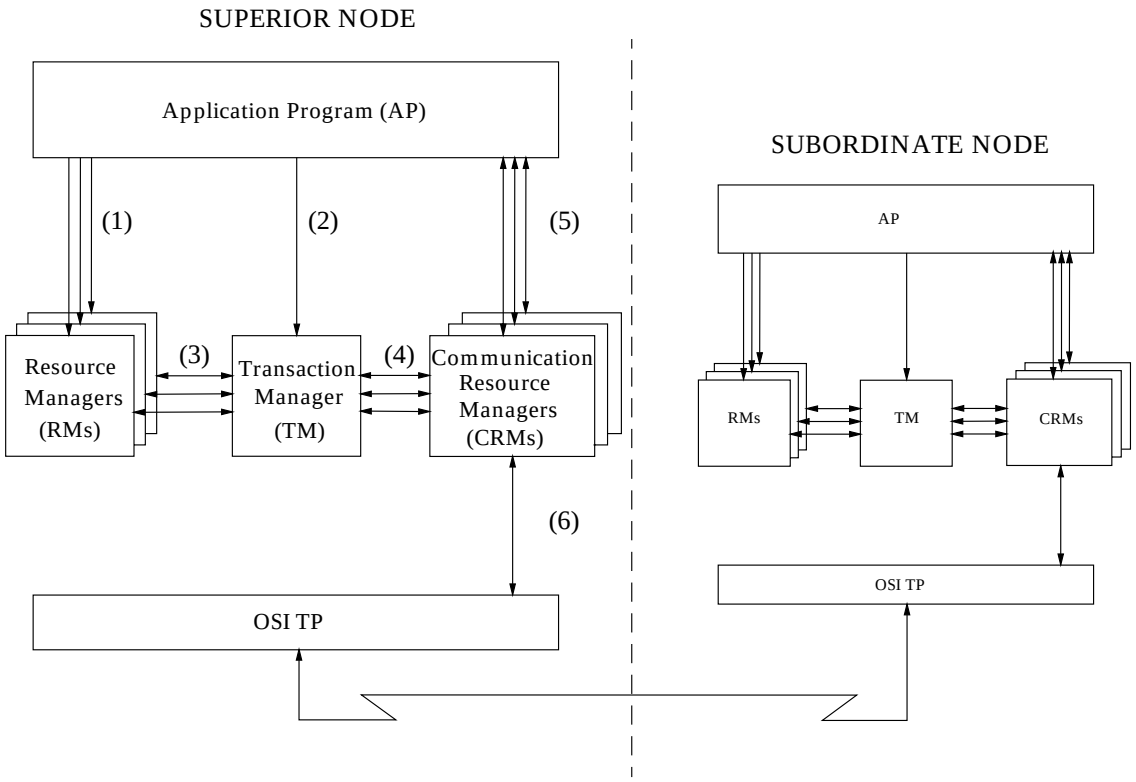


Figure 2-1 Functional Components and Interfaces

Descriptions of the functional components shown can be found in Section 2.1.1 on page 6. The numbers in brackets in the above figure represent the different X/Open interfaces that are used in the model. They are described in Section 2.1.2 on page 7.

For more details of this model and diagram, including detailed definitions of each component, see the referenced **DTP** guide.

2.1.1 Functional Components

Application Program (AP)

The application program (AP) implements the desired function of the end-user enterprise. Each AP specifies a sequence of operations that involves resources such as databases. An AP defines the start and end of global transactions, accesses resources within transaction boundaries, and normally makes the decision whether to commit or roll back each transaction.

Where two or more APs cooperate within a global transaction, the X/Open DTP model supports three *paradigms* for AP to AP communication. These are the TxRPC, XATMI and CPI-C interfaces.

Transaction Manager (TM)

The transaction manager (TM) manages global transactions and coordinates the decision to start them, and commit them or roll them back. This ensures atomic transaction completion. The TM also coordinates recovery activities of the resource managers when necessary, such as after a component fails.

Resource Manager (RM)

The resource manager (RM) manages a defined part of the computer's shared resources. These may be accessed using services that the RM provides. Examples for RMs are database management systems (DBMSs), a file access method such as X/Open ISAM, and a print server.

In the X/Open DTP model, RMs structure all changes to the resources they manage as recoverable and atomic transactions. They let the TM coordinate completion of these transactions atomically with work done by other RMs.

Communication Resource Manager (CRM)

A CRM allows an instance of the model to access another instance either inside or outside the current TM Domain. Within the X/Open DTP model, CRMs use OSI TP services to provide a communication layer across TM Domains. CRMs aid global transactions by supporting the following interfaces:

- the communication paradigm (TxRPC, XATMI or CPI-C) used between an AP and CRM
- XA+ communication between a TM and CRM
- XAP-TP communication between a CRM and OSI TP.

A CRM may support more than one type of communication paradigm, or a TM Domain may use different CRMs to support different paradigms. The XA+ interface provides global transaction information across different instances and TM Domains. The CRM allows a global transaction to extend to another TM Domain, and allows TMs to coordinate global transaction commit and abort requests from (usually) the superior AP. Using the above interfaces, information flows from superior to subordinate and *vice versa*.

2.1.2 Interfaces between Functional Components

There are six interfaces between software components in the X/Open DTP model. The numbers correspond to the numbers in Figure 2-1 on page 5.

(1) **AP-RM.** The AP-RM interfaces give the AP access to resources. X/Open interfaces, such as SQL and ISAM, provide AP portability. The X/Open DTP model imposes few constraints on native RM APIs. The constraints involve only those native RM interfaces that define transactions. (See the referenced **XA** specification.)

(2) **AP-TM.** The AP-TM interface (the TX interface) provides the AP with an Application Programming Interface (API) by which the AP coordinates global transaction management with the TM. For example, when the AP calls *tx_begin*( ) the TM informs the participating RMs of the start of a global transaction. After each request is completed, the TM provides a return value to the AP reporting back the success or otherwise of the TX call.

For details of the AP-TM interface, see the referenced **TX** (Transaction Demarcation) specification.

(3) **TM-RM.** The TM-RM interface (the XA interface) lets the TM structure the work of RMs into global transactions and coordinate completion or recovery. The XA interface is the bidirectional interface between the TM and RM.

The functions that each RM provides for the TM are called the *xa_**( ) functions. For example the TM calls *xa_start*( ) in each participating RM to start an RM-internal transaction as part of a new global transaction. Later, the TM may call in sequence *xa_end*( ) *xa_prepare*( ) and *xa_commit*( ) to coordinate a (successful in this case) two-phase commit protocol. The functions that the TM provides for each RM are called the *ax_**( ) functions. For example an RM calls *ax_reg*( ) to register dynamically with the TM.

For details of the TM-RM interface, see the referenced **XA** specification.

(4) **TM-CRM.** The TM-CRM interface (the XA+ interface) supports global transaction information flow across TM Domains. In particular TMs can instruct CRMs by use of *xa_**( ) function calls to suspend or complete transaction branches, and to propagate global transaction commitment protocols to other transaction branches. CRMs pass information to TMs in subordinate branches by use of *ax_**( ) function calls. CRMs also use *ax_**( ) function calls to request the TM to create subordinate transaction branches, to save and retrieve recovery information, and to inform the TM of the start and end of blocking conditions.

For details of the TM-CRM interface, see the referenced **XA+** specification.

The XA+ interface is a superset of the XA interface and supersedes its purpose. Since the XA+ interface is invisible to the AP, the TM and CRM may use other methods to interconnect without affecting application portability.

(5) **AP-CRM.** X/Open provides portable APIs for DTP communication between APs within a global transaction. The API chosen can significantly influence (and may indeed be fundamental to) the whole architecture of the application. For this reason, these APIs are frequently referred to in this specification and elsewhere as *communication paradigms*. In practice, each paradigm has unique strengths, so X/Open offers the following popular paradigms:

- the TxRPC interface (this document)
- the XATMI interface (see the **XATMI** specification)
- the CPI-C interface (see the **CPI-C** specification).

X/Open interfaces, such as the three CRM APIs listed above, provide application portability across products offering the same CRM API. The X/Open DTP model imposes few constraints on native CRM APIs.

(6) **CRM-OSI TP.** This interface (the XAP-TP interface) provides a programming interface between a CRM and Open Systems Interconnection Distributed Transaction Processing (OSI TP) services. XAP-TP interfaces with the OSI TP Service and the Presentation Layer of the seven-layer OSI model. X/Open has defined this interface to support portable implementations of application-specific OSI services. The use of OSI TP is mandatory for communication between heterogeneous TM domains. For details of this interface, see the referenced **XAP-TP** specification and OSI TP standards.

2.2 Definitions

For additional definitions see the referenced **DTP** guide.

2.2.1 Transaction

A transaction is a complete unit of work. It may comprise many computational tasks, which may include user interface, data retrieval, and communication. A typical transaction modifies shared resources. (The OSI TP standards (model) defines transactions more precisely.)

Transactions must be able to be *rolled back*. A human user may roll back the transaction in response to a real-world event, such as a customer decision. A program can elect to roll back a transaction. For example, account number verification may fail or the account may fail a test of its balance. Transactions also roll back if a component of the system fails, keeping it from retrieving, communicating, or storing data. Every DTP software component subject to transaction control must be able to undo its work in a transaction that is rolled back at any time.

When the system determines that a transaction can complete without failure of any kind, it *commits* the transaction. This means that changes to shared resources take permanent effect. Either commitment or rollback results in a consistent state. *Completion* means either commitment or rollback.

2.2.2 Transaction Properties

Transactions typically exhibit the following properties:

Atomicity The results of the transaction's execution are either all committed or all rolled back.

Consistency A completed transaction transforms a shared resource from one valid state to another valid state.

Isolation Changes to shared resources that a transaction effects do not become visible outside the transaction until the transaction commits.

Durability The changes that result from transaction commitment survive subsequent system or media failures.

These properties are known by their initials as the **ACID** properties. In the X/Open DTP model, the TM coordinates Atomicity at global level whilst each RM is responsible for the Atomicity, Consistency, Isolation and Durability of its resources.

2.2.3 Distributed Transaction Processing

Within the scope of this document, DTP systems are those where work in support of a single transaction may occur across RMs. This has several implications:

- The system must have a way to refer to a transaction that encompasses all work done anywhere in the system.
- The decision to commit or roll back a transaction must consider the status of work done anywhere on behalf of the transaction. The decision must have uniform effect throughout the DTP system.

Even though an RM may have an X/Open-compliant interface such as Structured Query Language (SQL), it must also address these two items to be useful in the DTP environment.

2.2.4 Global Transactions

Every RM in the DTP environment must support transactions as described in Section 2.2.1 on page 9. Many RMs already structure their work into recoverable units.

In the DTP environment, many RMs may operate in support of the same unit of work. This unit of work is a *global transaction*. For example, an AP might request updates to several different databases. Work occurring anywhere in the system must be committed atomically. Each RM must let the TM coordinate the RM's recoverable units of work that are part of a global transaction.

Commitment of an RM's internal work depends not only on whether its own operations can succeed, but also on operations occurring at other RMs, perhaps remotely. If any operation fails anywhere, every participating RM must roll back all operations it did on behalf of the global transaction. A given RM is typically unaware of the work that other RMs are doing. A TM informs each RM of the existence, and directs the completion, of global transactions. An RM is responsible for mapping its recoverable units of work to the global transaction.

2.2.5 Transaction Branches

A global transaction has one or more *transaction branches* (or *branches*). A branch is a part of the work in support of a global transaction for which the TM and the RM engage in a separate but coordinated transaction commitment protocol. Each of the RM's internal units of work in support of a global transaction is part of exactly one branch.

A global transaction might have more than one branch when, for example, the AP uses a CRM to communicate with remote APs. The CRM asks the TM to create a new transaction branch prior to accessing a remote AP for the first time. Subsequent accesses to the same remote AP are typically done within the same transaction branch. Accesses to different remote APs are typically done in separate transaction branches.

After the TM begins the transaction commitment protocol, the RM receives no additional work to do on that transaction branch. The RM may receive additional work on behalf of the same transaction, from different branches. The different branches are related in that they must be completed atomically. However, the TM directs the commitment protocol for each branch separately. That is, an RM receives a separate commitment request for each branch.

2.2.6 Remote Procedure Call

A *remote procedure call* (RPC) is a programming paradigm similar to the well-known procedure call mechanism. Both procedure call mechanisms transfer control and data within a program.

When a remote procedure is called, the parameters of the call are passed over the network to the environment where the call is actually executed. Meanwhile, the calling environment waits for the results of the procedure execution. Typically the calling environment is a program that is referred to as a *client*. The environment where the call is executed is referred to as a *server*. When the server (the called environment) finishes executing the procedure, it returns the results back to the client (the calling environment) which then resumes execution as if returning from a local procedure call.

2.2.7 Client and Server

A *client* is a program that issues remote procedure calls; a *server* is a program that accepts remote procedure calls. A program may be both a server and a client. The *root client* is the client that initiated the transaction by calling *tx_begin*( ).

2.2.8 Manager Function

A *manager function* is the application procedure executed in a server that implements an operation. A manager function is a part of the AP in the X/Open Distributed Transaction Processing (DTP) Model.

2.2.9 Transactional RPC

Transactional RPCs are RPCs that are executed within the scope of a global transaction. The transactional context of the caller is automatically communicated to the server. The manager function in the server executes within the scope of the caller's transaction. Therefore, any work performed by the manager function with RMs is contained within the same global transaction as the caller.

2.2.10 Interface Definition Language

An *Interface Definition Language* (IDL) is a language for specifying operations (procedures and functions), parameters to these operations and data types. An operation description includes whether an operation must be executed as a transactional RPC, may or may not be executed as a transactional RPC, or must not be executed as a transactional RPC. For a complete definition of the DCE IDL, read Section 4.2, IDL Language Specification, of the X/Open **DCE RPC** specification.

2.2.11 TxRPC Communication Resource Manager

Section 2.1.1 on page 6 discusses the concept of a CRM. A TxRPC CRM is a CRM that uses the interface specified herein to allow an AP to communicate with other APs using a remote procedure call communication paradigm. These communication operations may propagate a global transaction to other APs involved in the communication.

There are two types of TxRPC CRM, an IDL-only TxRPC CRM and an RPC TxRPC CRM. The phrase TxRPC CRM is used to describe functions and features applicable to both; otherwise the explicit reference is used.

2.3 TxRPC Model

The X/Open **DCE RPC** specification defines a remote procedure call facility. RPC provides non-transactional RPCs for use by an AP. The RPC model specifies the mechanisms by which a client locates a server and invokes a manager function. This model is described in Section 6.1, Client/Server Execution Model, of the X/Open **DCE RPC** specification.

The TxRPC CRM includes and enhances RPC functionality. The TxRPC CRM permits the AP to extend the implied context of a manager function to include the global transaction that the client is working on at the time of the RPC. The API provided by the TxRPC CRM is a modified version of the API in the X/Open **DCE RPC** specification. Specifically, some additional operator and interface attributes within IDL are specified, the use of some operator and interface attributes are restricted, and some parameter constructions used in operation specification are restricted. These extensions and restrictions are specified in Chapter 3.

Operations that have either the **transaction_mandatory** or **transaction_optional** attributes applied to them are called TxRPC operations, and calls to the corresponding manager functions are defined to be TxRPCs. If a TxRPC takes place within the scope of a transaction, the call is defined to be a transactional RPC.

Note: Not all TxRPCs are transactional RPCs. In particular, operations with the **transaction_optional** attribute that are not invoked within the scope of a transaction are not transactional RPCs.

2.4 Transaction Implications

The TxRPC CRM relies on the X/Open **TX** (Transaction Demarcation) interface, published separately, for global transaction demarcation and management. In addition, certain functions in the TxRPC interface directly affect the progress of the global transaction.

2.4.1 Transaction Functions Affecting a TxRPC CRM

Demarcation

The TxRPC CRM interface relies on the following functions of the Transaction Demarcation (TX) interface:

tx_begin( ) A demarcation function that indicates that subsequent work performed by the calling application thread of control is in support of a global transaction.

tx_commit( ) A demarcation function that commits all work done on behalf of the current global transaction.

tx_rollback( ) A demarcation function that indicates an AP's desire to roll back all work done on behalf of the current global transaction.

Timeout

The timeout function of TX also affects the TxRPC interface:

tx_set_transaction_timeout( )
A function that specifies the time interval in which the transaction must complete.

Chaining

The TxRPC interface is also affected by transaction chaining:

tx_set_transaction_control( )
A function to set the AP to chained or unchained mode.

Information

The AP can determine if it is executing within the scope of a transaction by using the following function:

tx_info( ) A function that returns global transaction information.

2.4.2 TxRPCs in a Transactional Environment

TxRPCs are used in an AP in the same manner as RPCs. A TxRPC becomes a transactional RPC if its operation is specified as transaction-optional or transaction-mandatory and the call is invoked within the scope of a transaction. A transaction normally proceeds as follows:

1. Initiating the global transaction: the AP calls the *tx_begin*( ) function to initiate a global transaction.
2. Communicating via TxRPCs: the AP makes a sequence of transactional remote procedure calls to servers which cause manager functions to be executed as part of the global transaction.
3. Completing the global transaction: the AP calls the *tx_commit*( ) function to commit the global transaction.

Under normal circumstances, the global transaction would be committed.

2.5 Transaction Commitment

The request to commit a transaction can only be made by the same AP thread that initiated the transaction. More specifically, this means that the AP thread that calls *tx_commit*( ) must be the same AP thread that called *tx_begin*( ).

2.6 Nested TxRPCs

An AP may be both a server and a client. For example, a manager function that is invoked by a client via a transactional RPC may in turn make its own transactional RPC to another server, thereby invoking another manager function. The second server is in the same global transaction as the first server. The global transaction extends to work done by any sequence of nested transactional RPCs initiated from a client.

Any server that is invoked by a client on behalf of a global transaction, and that makes transactional RPCs to other servers on behalf of the same transaction, must complete those transactional RPCs before returning to the original client. The transaction initiator must likewise ensure that all transactional RPCs it made are completed before calling *tx_commit*( ).

2.7 Non-transactional RPCs

APs may make non-transactional RPCs in addition to transactional RPCs. Non-transactional RPCs do not propagate global transactions; work done during non-transactional RPCs is not contained in the scope of any transaction that may be active in the RPC caller.

There are two ways to invoke non-transactional RPCs. Firstly, the called operation is declared with no transactional attribute. Using this mechanism, there is never any transaction context communicated from the client to the server, even if the client is in a global transaction. Secondly, the operation is declared with the **transaction_optional** attribute and is called outside the scope of a global transaction. Again no transaction context is communicated from client to server.

For the purposes of this specification, the invoked manager function of a non-transactional RPC is executing as a client with respect to the TxRPC CRM. For example, the invoked manager function may initiate and complete transactions, and those transactions are affected by the setting of any transaction characteristics at the time of transaction initiation.

Although there are two ways for non-transactional RPCs to be invoked, the two operation specifications are not interchangeable. Specifically, an operation that is specified without any transaction attribute is not equivalent to an operation that has the **transaction_optional** attribute and whose manager function is invoked outside the scope of a transaction. Both client and server must use the same specification for a non-transactional RPC: either no transaction attribute or the **transaction_optional** attribute.

2.8 Transaction Rollback

Rollback occurs at the TM in response to a *tx_rollback*( ) call or for other reasons. When rollback occurs at the TM, the TM informs the TxRPC CRM of this event. Subsequent behaviour depends on whether rollback occurs in a root client or server.

When rollback occurs in the root client, the TM initiates a rollback which is propagated by the TxRPC CRM. When all subordinates have been successfully rolled back, the client TxRPC CRM notifies the TM of this fact.

When rollback occurs in the server TM (such as by a call of *tx_rollback*( ) by the manager function), the TxRPC CRM marks the transaction *rollback-only*. This information is propagated to the client TM when the manager function returns results. After the results have been returned, the client remains in transaction mode and can still make TxRPC calls. However, any work performed on behalf of the transaction is ultimately rolled back[1].

It is recommended that in cases where the manager function calls *tx_rollback*( ) it also returns an application-specific error to indicate that the transaction must be rolled back. In such a case the client would normally call *tx_rollback*( ) after receiving results. However, the client is not required to call *tx_rollback*( ).

An intermediate node is both a client and a server at the same time. As the TxRPC CRM in a server is not allowed to initiate a rollback unilaterally, if the AP issues *tx_rollback*( ), the TxRPC CRM marks the transaction rollback-only. This information is returned to the client when the server returns, with the call results according to the rules above.

1. The state of the transaction is available to the AP by use of the *tx_info*( ) service.

Chapter 3

Interface Overview

This chapter gives an overview of the TxRPC interface and describes its relationship to the TX interface. In an X/Open DTP system, TxRPC is the interface between an AP and a CRM, and TX is the interface between an AP and a TM.

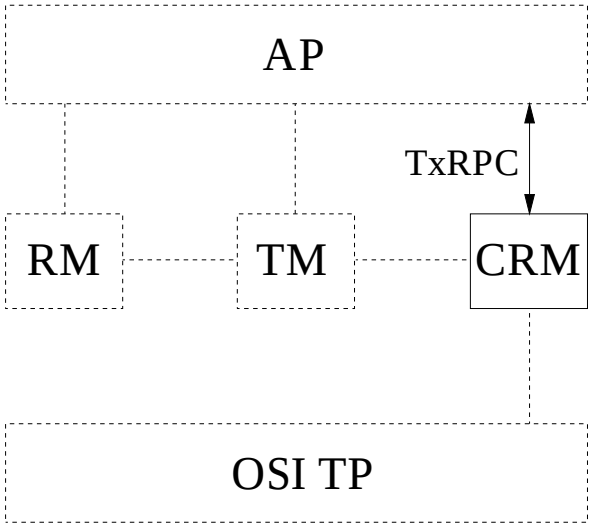


Figure 3-1 The TxRPC Interface

The TxRPC CRM application programming interface is an extension of, and defined in terms of, two other interfaces that are defined in the X/Open **DCE RPC** specification and the **TX** (Transaction Demarcation) specification.

The communication part of the API for the TxRPC CRM is based on the X/Open **DCE RPC** specification, which is composed of three parts: the specification of the IDL (Interface Definition Language), programs involved in creating and processing an IDL file, and a collection of routines that may be called from an AP. Two of these are modified for this specification. Firstly, IDL is both restricted and expanded. The restrictions remove capabilities that are not well suited to a transactional environment, while the additions are used to specify transactional properties of operations. Secondly, for an RPC TxRPC CRM, some new parameter values for run-time procedures specified in the X/Open **DCE RPC** specification must be made available.

The transaction management part of the API for the TxRPC CRM is based on the **TX** (Transaction Demarcation) specification. The **TX** (Transaction Demarcation) specification specifies behaviour only when called from a client AP (that is, the root of a transaction tree). However, APs that use the TxRPC CRM interface may be servers (that is, intermediate or leaf nodes of a transaction tree). Therefore, this document also discusses how the interface specified in the **TX** (Transaction Demarcation) specification can be used within a server AP.

This chapter discusses the changes to the X/Open **DCE RPC** specification, and the relationship to the **TX** (Transaction Demarcation) specification for the TxRPC CRM.

3.1 Interactions with the RPC Interface

3.1.1 IDL Language Interactions

The nature of a TxRPC CRM places limitations on the RPC options as specified in Section 4.2, IDL Language Specification, of the X/Open **DCE RPC** specification. The sections below document the ways that an RPC may be specified in order to be used by a TxRPC CRM. In particular, the following two sections discuss which additional features are available and which features of IDL must not be used in a specification for TxRPC calls.

3.1.2 Additional IDL Attributes

Operations that may carry transaction information must be identified in the IDL file through the use of attributes defined in this TxRPC specification. Two mutually exclusive attributes are available: **transaction_mandatory** and **transaction_optional**. These attributes may appear anywhere that other IDL attributes are permitted in an IDL file. If an interface is specified as either transaction-mandatory or transaction-optional, neither attribute may be specified for any individual operation in the IDL file.

Operations that are specified as being transaction-mandatory may be invoked only within the scope of a global transaction. When invoked, the work performed by the manager function is included in the scope of the global transaction of the caller. If an attempt is made to invoke a transaction-mandatory operation outside the scope of the caller's transaction, the **txrpc_s_not_in_transaction** status code is returned to the client and the manager function is not executed.

Operations that are specified as being transaction-optional may be invoked within the scope of a global transaction or outside a global transaction. When invoked within the scope of a global transaction, the work performed by the manager function is included in the scope of the global transaction of the caller. If invoked outside the scope of a transaction, the manager function does not begin execution within the scope of any transaction. The manager function may, however, initiate a transaction during its execution.

3.1.3 Limiting IDL Attributes

Transactional communication works best in an environment where each interaction can be positively verified. Interactions with anonymous, unresponsive or changing recipients are not consistent with transactional guarantees. Therefore, the following attributes (as defined in Section 4.2.22, Operations, of the X/Open **DCE RPC** specification) may not be specified for a TxRPC operation:

- **broadcast**
- **maybe**
- **idempotent**.

The following structured type (as defined in Section 5.1.4, Pipes, of the X/Open **DCE RPC** specification) is not supported for TxRPC operations:

- **pipe**.

3.1.4 Context Handles

Context handles (as defined in Section 4.2.16.6, The context_handle Attribute, of the X/Open **DCE RPC** specification) are user-defined data structures that hold data on a server between calls of a client to that server. Context handles for a TxRPC CRM are declared, allocated on the server side and interrogated on the client side with TxRPC just as with the unenhanced RPC mechanisms. A context handle can be created in either a transactional or non-transactional call, and may subsequently be used with both transactional and non-transactional operations. A context handle used within a transaction may persist beyond the life of the transaction and may subsequently be used within a new transaction.

The lifetime of the context handle is under the control of the application writer. A context handle may be destroyed by the server as part of either a transactional or non-transactional operation. Destroying a context handle has no effect on any current transaction. In the case of communication failure however, the RPC run-time system may reclaim context handles. Server APs are informed of this circumstance by having the associated context-handle run-down procedures executed. Context run-down procedures are always invoked outside the scope of a transaction.

Context handles are not bound to an interface. A context handle may be used with an operation in any interface supported by the server that creates the handle.

Context handles do not refer to explicit transaction states in their user-defined data structures. Like all APs, a TxRPC server AP can get information about the transactional state of the thread by calling *tx_info*( ).

The particular semantics of context handles when used in TxRPC calls are defined as follows:

- Multiple operations may only be guaranteed to execute in the same server address space via use of a valid context handle. This is an example of where the behaviour of TxRPC context handles differs from unenhanced RPC.

3.1.5 OSI TP Protocol Sequence

Interoperable TxRPC CRMs use the OSI TP protocols as described in this specification. The OSI TP protocols assume an OSI transport and an OSI addressing mechanism. The binding services of the RPC run-time environment must be able to select the OSI protocol sequence instead of the TCP/IP protocol sequence. Therefore, an interoperable implementation of an RPC TxRPC CRM must support the new predefined string **ncacn_osi_tp** as a selector for the OSI TP protocol sequence. This string is not needed by the AP in the case of an IDL-only TxRPC CRM (the stack should be defined by the environment). The **ncacn_osi_tp** string can be used as a parameter to the unenhanced RPC run-time functions that select protocol sequences.

3.1.6 RPC Run-time Service Interactions

The function *tx_open*( ) must be called before a server's TxRPC manager function is invoked on behalf of a global transaction; the manner in which *tx_open*( ) is called is implementation-dependent. The function *tx_open*( ) need not be called before invoking TxRPC manager functions invoked on behalf of no global transaction. There is no requirement that *tx_open*( ) be called by a client invoking any non-transactional RPCs.

If *tx_open*( ) is not called at the server before the manager function of a TxRPC operation is invoked on behalf of a global transaction, the **txrpc_s_no_tx_open_done** status code is returned to the client.

3.1.7 IDL-only TxRPC CRMs

IDL-only TxRPC CRMs support all language bindings derived from the IDL with the exception of the **handle_t** type and the **handle** attribute. Further, IDL-only TxRPC CRMs need not provide the programs or run-time routines specified in the X/Open **DCE RPC** specification.

3.1.8 TxRPC Errors

The following status codes may be returned to the client by a TxRPC CRM:

- **txrpc_s_not_in_transaction**

  A transaction-mandatory operation was invoked outside the scope of the caller's global transaction.

- **txrpc_s_no_tx_open_done**

  An operation was invoked but *tx_open*( ) was not called at the TxRPC server prior to the manager function's invocation.

Return of these status codes is required for compliant TxRPC CRM implementations. Optionally, a TxRPC CRM may also support return of these errors by raising exceptions. The corresponding exception codes which would be raised at the client are:

- **txrpc_x_not_in_transaction**
- **txrpc_x_no_tx_open_done**.

3.1.9 Object Support

As in unenhanced RPC, an object UUID can be associated with a TxRPC operation. The object UUID can be used, in addition to the interface UUID and version, to affect selection of the server AP. It can also be used within a server to support multiple implementations (base on object type) of an interface.

3.2 Interactions with the TX Interface

The following discussion of the interactions between a TxRPC CRM and the TX interface is made under the assumption that a client is calling the *tx_**()* functions. However, a server may act as a client if it is communicating with another server using TxRPC.

3.2.1 tx_begin()

The *tx_begin*() function initiates a global transaction. The AP that calls *tx_begin*() becomes the transaction initiator. A manager function called within the scope of a transaction must not call *tx_begin*().

3.2.2 tx_close()

The *tx_close*() function may be called by either the client or server in states defined by the **TX** (Transaction Demarcation) specification.

3.2.3 tx_commit()

The *tx_commit*() function may be called only by the transaction initiator.

3.2.4 tx_info()

The *tx_info*() function may be called by either the client or server in states defined by the **TX** (Transaction Demarcation) specification.

3.2.5 tx_open()

The *tx_open*() function may be called by either the client or server in states defined by the **TX** (Transaction Demarcation) specification. However, a *tx_open*() call must have been made by a server before it can receive any TxRPC calls. In a client, *tx_open*() must be called before any TxRPC calls are made.

3.2.6 tx_rollback()

The *tx_rollback*() function may be called by either the client or server. When a manager function calls *tx_rollback*(), the global transaction is marked rollback-only. See Section 2.8 on page 15 for a discussion of transaction rollbacks in a TxRPC environment.

3.2.7 tx_set_commit_return()

The *tx_set_commit_return*() function may be called by either the client or server. The effect of this call only pertains to transactions for which the AP is the transaction initiator. In particular, if the call is made while a manager function is being called on behalf of a transactional RPC, the value is remembered by the TM, but does not apply to the current transaction, or to the AP that initiated the current transaction.

3.2.8 tx_set_transaction_control()

The *tx_set_transaction_control*() function may be called by either the client or server. The effect of this call only pertains to transactions for which the AP is the transaction initiator. In particular, if the call is made while a manager function is being called on behalf of a transactional RPC, the value is remembered by the TM, but does not apply to the current transaction, or to the AP that initiated the current transaction.

3.2.9 tx_set_transaction_timeout()

The *tx_set_transaction_timeout*() function may be called by either the client or server. The effect of this call only pertains to transactions for which the AP is the transaction initiator. In particular, if the call is made while a manager function is being called on behalf of a transactional RPC, the value is remembered by the TM, but does not apply to the current transaction, or to the AP that initiated the current transaction.

Chapter 4

Implementation Requirements

This chapter summarises the implications for implementors of the TxRPC CRM. It also identifies features of this specification that implementors of a TxRPC CRM or application writers can regard as optional.

These requirements are designed to facilitate portability: specifically, the ability to move an application program to a different X/Open DTP system without modifying the source code. It is anticipated that DTP products will be delivered as object modules and that the administrator will control the mix and operation of components at a particular site by doing one or more of the following:

- relinking object modules
- supplying text strings to the software components (or executing a vendor supplied procedure that incorporates suitable text strings).

4.1 AP Requirements

Any AP in a DTP system must use a TM and delegate to it responsibility to control and coordinate each global transaction.

The AP is not involved in either the commitment protocol or the recovery process. An AP thread of control can have only one global transaction active at a time.

4.2 Thread of Control

It is important that the AP, TM, RMs, and CRMs agree on the definition of a thread of control. The Transaction Processing Profile in the **XDCS** guide specifies that the operating system, threading, communication and remote procedure call components be present. The threading component of the **XDCS** guide allows for multiple threads within a single operating system process. However, the **XA** specification defines thread of control to be an operating system process. Special considerations may need to be made when constructing your DTP system. For a further discussion of threads, see Section 6.1.7, Threads, of the X/Open **DCE RPC** specification.

4.3 TxRPC CRM Requirements

4.3.1 Compliant TxRPC CRMs

An implementor can choose to offer either or both of the two types of compliant TxRPC CRM:

- RPC TxRPC CRM

  The XDCE RPC API, IDL, and Stubs are supported as defined by the following parts of the referenced X/Open **DCE RPC** specification, except as modified by this (**TxRPC**) specification:

  — Part 2: RPC Application Programmer's Interface

  — Part 3: Interface Definition Language and Stubs.

- IDL-only TxRPC CRM

  The XDCE IDL only is supported as defined by the following chapter of the referenced X/Open **DCE RPC** specification, except as modified by this (**TxRPC**) specification:

  — Chapter 4: Interface Definition Language.

  No XDCE RPC API run-time services are provided; default parameters are used by the stubs. With the exception of Section 3.1.5 on page 19, all sections within Section 3.1 of this (**TxRPC**) specification are applicable to IDL-only TxRPC CRMs.

Unless otherwise specified, all sections in this document are applicable to both RPC and IDL-only TxRPC CRMs. These two types of CRM interoperate if the applications using the RPC TxRPC CRM only use *interface definitions* that have been specified in the IDL subset permitted to IDL-only applications.

4.3.2 Public Information

The X/Open TxRPC CRM must specify:

- the names of libraries or objects files, in the correct sequence, that the administrator must use when linking applications with the X/Open TxRPC CRM
- compiler names and options to build transactional RPC interfaces
- initialisation of the RPC run-time system (this only applies to the RPC TxRPC CRM)
- special considerations needed to reconcile the thread model
- whether or not the TxRPC CRM is IDL-only.

4.4 TM Requirements

TMs must support interaction with the X/Open TxRPC CRM.

TMs do not communicate with each other directly. They rely on the TxRPC CRM for transaction propagation and the propagation of the transaction completion protocol.

X/Open CAE Specification

Part 2:

TxRPC Application Service Element (ASE)

X/Open Company Ltd.

Chapter 5

Remote Task Invocation Model

This chapter describes the basic Remote Task Invocation (RTI) Model and the RTI Communication Model. It also compares the RTI Model with the OSI TP Model and OSI Application Layer Structure (ALS).

The basic structure of the RTI Model is derived from the OSI Application Layer Structure, described in ISO/IEC 9545, and the ISO/IEC Distributed Transaction Processing standard, described in the OSI TP Model, Service and Protocol standards.

5.1 Model Components

The four primary components of the RTI Model are the RTI Application Process Invocation, the RTI Service User Invocation, the RTI Application Entity Invocation, and the RTI Protocol Machine. Each RTI Application Process Invocation may have many RTI Service User Invocations, each with its associated RTI Protocol Machine.

Figure 5-1 on page 29 depicts these components (and their sub-components) and shows how they relate to each other. Section 5.4 on page 39 relates the components of the RTI Model to the OSI TP Model and OSI Application Layer Structure.

5.1.1 RTI Application Process Invocation

An RTI Application Process Invocation (RTI-API) is an instance of an RTI Application Process (RTI-AP).

5.1.2 RTI Service User Invocation

An RTI Service User Invocation (RTI-SUI) is a user of the RTI Service provided by an RTI-PM. There is exactly one RTI-PM for each RTI-SUI. The service boundary between an RTI-SUI and an RTI-PM is referred to as the RTI Service Boundary. This boundary is invariant to an RTI-SUI regardless of the underlying communication protocol or protocols used by an RTI-PM. This allows an RTI-PM to select the appropriate underlying communication protocol or protocols without directly impacting its RTI-SUI.

5.1.3 RTI Application Entity Invocation

An RTI Application Entity Invocation (RTI-AEI) is an instance of an RTI Application Entity (RTI-AE). The RTI-AE is the subcomponent of an RTI-AP that is responsible for providing RTI based communication services.

5.1.4 RTI Protocol Machine

An RTI Protocol Machine (RTI-PM) contains a complete OSI TP Protocol Machine (TPPM), integrating both the OSI TP service and the OSF Remote Procedure Call (OSF RPC) data transfer service. The two primary components of an RTI-PM are the RTI Multiple Association Control Function (RTI-MACF) and the Single Association Object (SAO). An RTI-PM can have multiple SAOs, however only one SAO is used by an RTI-MACF at any one time. Each association being used by an RTI-PM is represented by an SAO. The RTI-PM is described in Chapter 8.

5.1.5 RTI Multiple Association Control Function

The RTI Multiple Association Control Function (RTI-MACF) coordinates all cross-association functions within an RTI-PM. In particular, the RTI-MACF ensures the proper sequencing of Application Protocol Data Units (APDUs) across associations. These functions form a set of multiple association coordination and sequencing rules that are called MACF rules.

The RTI-MACF contains the TP MACF which provides the OSI TP service to an RTI-PM. The RTI-MACF, in turn, provides the RTI service primitives to an RTI-PM. The RTI-MACF is an augmentation of the TP MACF and an RTI-PM is an augmentation of a TPPM.

5.1.6 Single Association Object

Each Single Association Object (SAO) contains one Single Association Control Function (SACF) and a number of Application Service Elements (ASEs) that support RTI-SUI-specific communication, and the single association aspects of an RTI-PM.

5.1.7 Single Association Control Function

The SACF is responsible for coordinating the use of a single association by multiple ASEs, and for the sequencing of outgoing APDUs (generated by the ASEs) on that association. For incoming APDUs, the SACF ensures delivery of the APDUs to the correct ASE. In general, the SACF coordinates the use of the Presentation Service by the individual components of an SAO. The SACF also increases communication efficiency by concatenating APDUs from one or more ASEs into a single Presentation Service Data Unit (PSDU). These functions form a set of single association coordination and sequencing rules that are called SACF rules.

In the RTI Model, the SACF coordinates the following ASEs:

- OSI Association Control Service Element (ACSE)
- OSI TP Application Service Element (TP-ASE)
- OSI Commitment, Concurrency and Recovery Application Service Element (CCR-ASE)
- Dialogue Control Application Service Element (DC-ASE)
- OSF Remote Procedure Call Application Service Element (RPC-ASE).

It should be noted that a CCR-ASE is only included in an SAO when data transfer with transaction semantics is required by an RTI-SUI.

An RTI-PM integrates all the components described above and maps the service primitives provided by the RTI-MACF to the RTI Service.

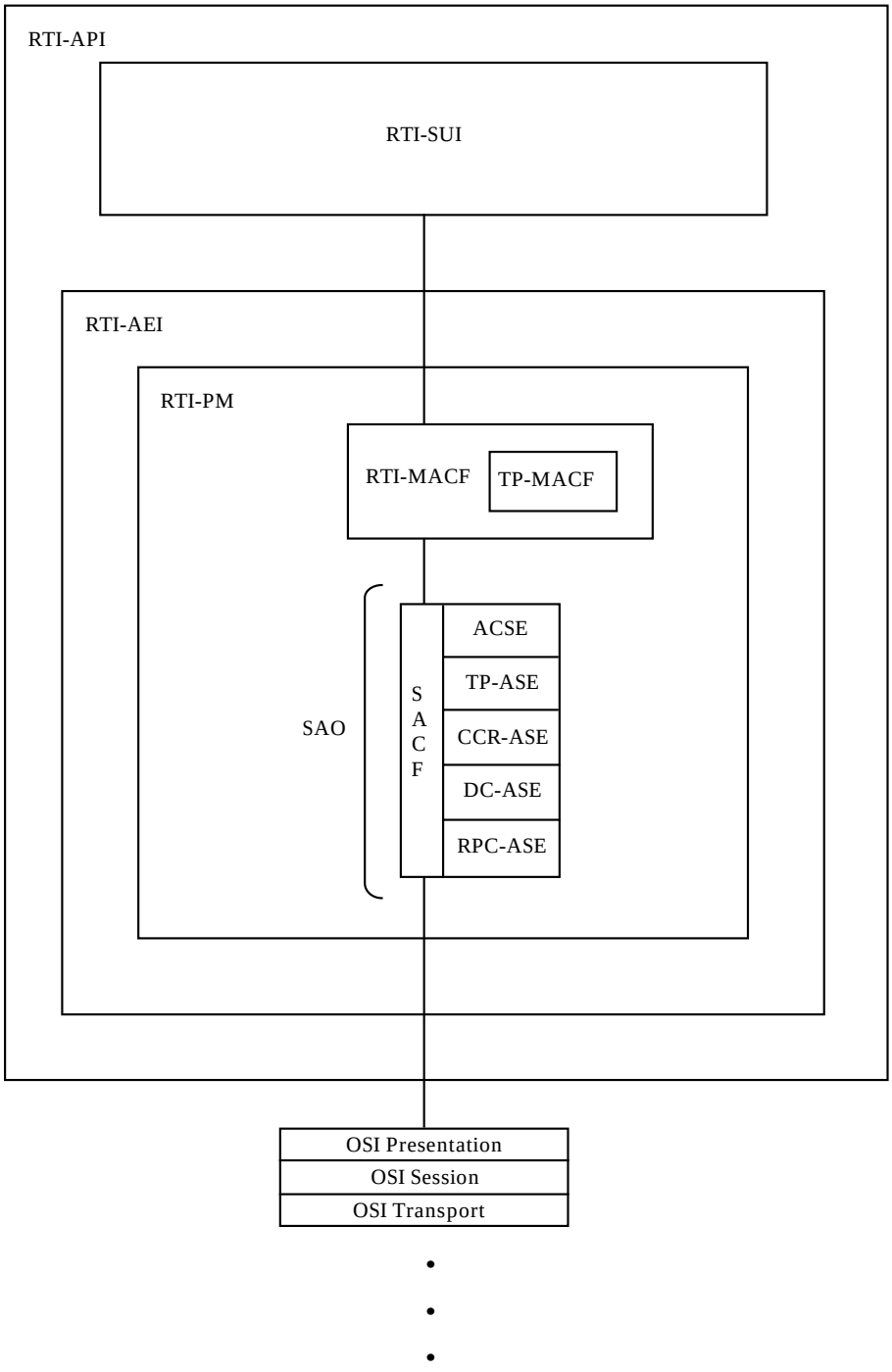


Figure 5-1 RTI Model

5.2 RTI Model Component Relationships

Each component of the RTI Model defines a set of abstract services to describe the structure, encoding and sequencing of APDUs. The relationship of the RTI Model components to each other and to external architectures (that is, OSI TP and OSF RPC) can be expressed in terms of how the abstract services provided by these components are mapped together to form the RTI Service.

The RTI Model is an OSI Application Layer model. At the highest level of the RTI Model is the RTI Service, and at the lowest level are the services provided by the OSI Presentation Layer. The following discussion describes the RTI Application Layer structure. That is, the various service primitives and services necessary to map the RTI Service to the OSI Presentation Layer. The mapping or layering of these service primitives is shown in Figure 5-2 on page 31 and described in Section 8.1.1 on page 82.

5.2.1 RPC-ASE Service Primitives

The service primitives that handle the OSF RPC data transfer for calls are modelled by the RPC-ASE. The RPC-ASE provides the OSF RPC service primitives to an RTI-PM. The RPC-ASE is an abstract model of the OSF RPC service and protocol integrated into the OSI TP environment for use by an RTI-AEI. The RPC-ASE is described in Section 8.4 on page 95 and the OSF RPC service and protocol are described in the referenced X/Open **DCE RPC** specification.

5.2.2 DC-ASE Service Primitives

The service primitives that handle the encoding of RTI-PM data (as opposed to RTI-SUI data) necessary for the proper operation of the RTI protocol are modelled by the DC-ASE. The DC-ASE provides general RTI protocol encoding service primitives to an RTI-PM. The DC-ASE is described in Section 8.3 on page 89.

5.2.3 OSI TP Service Primitives

The OSI TP service primitives are modelled as being provided by the TP MACF component of a TPPM. The TP MACF and the OSI TP service primitives are described in the OSI TP Model, Service and Protocol standards.

The TP MACF makes use of the service primitives provided by the following ASEs:

1. TP Application Service Element (TP-ASE) — described in the OSI TP Model, Service and Protocol standards
2. Association Control Service Element (ACSE) — described in ISO/IEC 8649 and 8650
3. OSI Commitment, Concurrency and Recovery Application Service Element (CCR-ASE) — described in ISO/IEC 9804 and 9805.

The service primitives provided by these three ASEs and their use by a TPPM to provide the OSI TP Service are described in the OSI TP Model, Service and Protocol standards.

5.2.4 RTI Service Primitives

The RTI service primitives are modelled as being provided by the RTI-MACF component of an RTI-PM. The RTI-MACF makes use of the service primitives provided by the RPC-ASE, the DC-ASE, and the TP MACF component of a TPPM (the OSI TP service primitives).

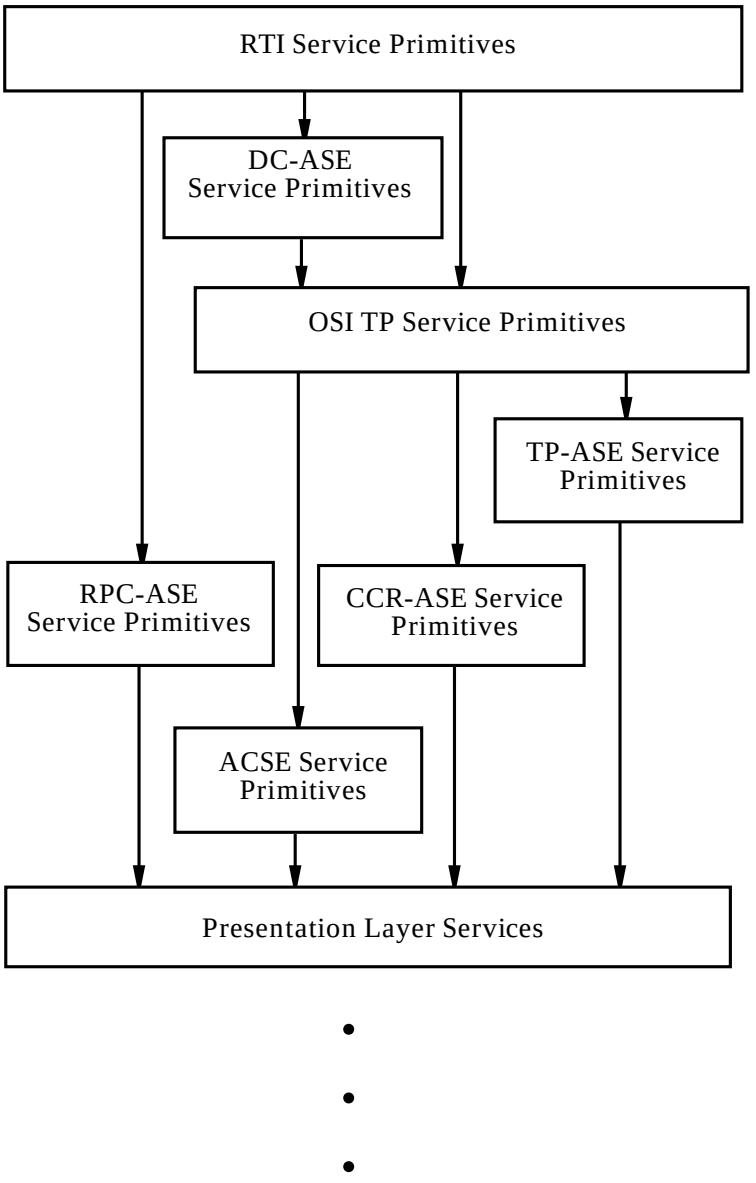


Figure 5-2 RTI Service Primitive Mapping (Abstract)

5.3 RTI Communication Model

5.3.1 Service Providers and Service Users

The term *service provider* describes an implementation of a particular service that accepts requests from, and issues indications to, a service user. The term *service user* describes an entity that issues requests to, and receives indications from, a service provider. In the RTI Model, the RTI Service Provider (RTI-SP) provides the RTI Service to all RTI-SUIs and embodies the RTI-PMs associated with those RTI-SUIs. The RTI-SP spans several RTI-APIs and is the conceptual view of the RTI Service as a whole. Any user of the RTI Service is referred to as an RTI Service User (RTI-SU). An RTI-SUI is an instance of an RTI Service User (RTI-SU). The service boundary between an RTI-SU and the RTI-SP is the RTI Service Boundary.

5.3.2 Clients and Servers

In the RTI Model a call is initiated by a *client* RTI-SUI. It is then processed by a *server* RTI-SUI, which returns the results to the client. When used without qualification, the terms *client* and *server* refer to a client RTI-SUI and a server RTI-SUI respectively. When used to qualify other RTI Model components, the terms *client* and *server* indicate that the particular component is associated with either a client RTI-SUI or server RTI-SUI respectively.

5.3.3 Processing a Call

A call is processed as follows:

1. A client issues a call request to its RTI-PM.
2. The client RTI-PM then sends the call request to a server RTI-PM which issues a call indication to a server.
3. When the server completes the processing associated with the call, it issues a call result request to its RTI-PM.
4. The server RTI-PM then sends the call result request to the client RTI-PM which issues a call result indication to the client.

If, at any time, the server is unable to process a call request, or when either a communication or system failure occurs, a call failure indication is returned to the client.

All call requests issued by a client are completed by the receipt of either a call result indication or a call failure indication.

All call indications received by a server must be completed by issuing a call result request, except when a rollback indication is received.

5.3.4 Contexts

Before issuing any call requests a client must first identify a server or servers that can process the requests for the client. To do this a client issues an *establish context* request to its RTI-PM. This request does not result in an immediate flow of protocol; rather it specifies a relationship between a client and a server. This relationship is referred to as a *context*[2]. All call requests issued within a particular context by a client are processed by the server identified by the context. There is no restriction on the number of calls that may be made within a particular context.

The client must maintain the relationship between each active context handle and the RTI context that was used for the RPC operation that created the context handle. This allows operations made with the context handle to be directed to the same server instance.

The type of context established specifies the characteristics of the relationship between a client and a server. A context may be one of two types; it is either *transaction-enabled* or *not transaction-enabled*. A context that is transaction-enabled can support both transactional and non-transactional operations. A context that is not transaction-enabled only supports non-transactional operations.

The context state is called *transactional* while it is associated with a transaction. This is from the time that the context is used for a transactional RPC until transaction termination (commit or rollback). At all other times the context state is said to be *non-transactional*.

The state of a context with transactions enabled can vary between transactional and non-transactional. The state of a context without transactions enabled is always non-transactional.

Each call is made using a context that identifies the server or servers that can process the request. New contexts are established as needed. A client can establish multiple contexts for the purpose of accessing multiple servers or to maintain multiple context handles with the same server. A context may be used to call operations in any interface supported by the server. A context with transactions enabled can be used for both transactional and non-transactional operations. A context without transactions enabled can only be used for non-transactional operations.

A context is established and maintained by an RTI-PM. A client considers a context to be established immediately after issuing an establish context request. A server considers a context to be established immediately upon receipt of a call made within the particular context.

Transactional contexts are maintained as long as the transaction is active. The context is automatically released at transaction termination (commit or rollback) if there are no active context handles that were created within that context. Otherwise, the context becomes non-transactional.

Non-transactional contexts are maintained while there are active context handles that were created within the context. The policy for releasing non-transactional contexts is implementation-dependent.

---

2. When the term *context* is used alone, it refers to an RTI context (created by an Establish Context request to an RTI-PM in the client, or by receipt of a Call Task indicator from the RTI-PM in the server). When referring to an RPC context, the term *context handle* is used.

5.3.5 Dialogues

When a client issues the first call request within a particular context to an RTI-PM (the client RTI-PM), the RTI-PM uses the information provided in the establish context request to establish a dialogue with a server RTI-PM. A dialogue may be one of two types; it is either *transaction-enabled* or *not transaction-enabled* A dialogue that is transaction-enabled is defined as an OSI TP dialogue on which the OSI TP Dialogue, Shared Control, Unchained Transaction, and Commit functional units have been selected. A dialogue that is not transaction-enabled is defined as an OSI TP dialogue on which only the OSI TP Dialogue and Shared Control functional units have been selected.

If a context is transaction-enabled, the underlying dialogue is created as transaction-enabled. The dialogue is created in unchained transaction mode allowing the dialogue to be used for both transactional and non-transactional RPCs.

If a context is not transaction-enabled, the underlying dialogue is created as not transaction-enabled. The dialogue is created in non-transaction mode, and can only be used for non-transactional RPCs.

The dialogue state is called *transactional* while it is associated with a transaction. This is from the time that the context is used for a transactional RPC until transaction termination (commit or rollback). At all other times the dialogue is said to be *non-transactional*.

The state of a dialogue with transactions enabled can vary between transactional and non-transactional. The state of a dialogue without transactions enabled is always non-transactional.

If a non-transactional dialogue is selected for a transactional RPC, the client RTI-PM explicitly requests to include the dialogue in the current transaction (by issuing a TP-BEGIN-TRANSACTION request on the dialogue).

Transactional dialogues are maintained as long as the transaction is active. The dialogue is automatically released at transaction termination (commit or rollback) if there are no active context handles that were created within that dialogue. Otherwise, the dialogue becomes non-transactional.

Non-transactional dialogues are maintained while there are active context handles that were created within the dialogue. The policy for releasing non-transactional dialogues is implementation-dependent.

Upon receipt of a *begin dialogue* indication, a server RTI-AEI creates an RTI-PM (the server RTI-PM) to accept the dialogue. The acceptance of the dialogue by the server RTI-PM completes the context establishment process. Once a dialogue is established between a client and a server RTI-PM the client RTI-PM can then send call requests and receive call result indications over that dialogue. Upon receipt of a call indication, a server RTI-PM then passes that indication to a server for processing.

If a non-transactional dialogue fails while a call is in progress, a *call failure* indication is issued by the client RTI-PM to the client. If a non-transactional dialogue fails when no call is in progress and there are no active context handles, a new dialogue is transparently re-established on the next call request as if it were the first call request issued within a context. If there are active context handles, a subsequent call request will fail with an error indicating the context handle is invalid. If a transactional dialogue fails while a call is in progress, a call failure indication followed by a rollback transaction indication is issued by the client RTI-PM to the client. If a transactional dialogue fails when no call is in progress, a rollback transaction indication is issued by the client RTI-PM to the client.

Dialogues are established and maintained by the RTI-PM. Transactional dialogues (without any active context handles) are automatically terminated by the RTI-PM upon transaction

termination (commit or rollback). When *tx_commit*( ) is invoked, all client RTI-PMs for transactional dialogues without active context handles set the dialogue to terminate with the transaction (deferred end dialogue). This occurs for all client RTI-PMs in the transaction tree. If the transaction aborts, the client RTI-PMs explicitly abort these dialogues. The termination policy for non-transactional dialogues is implementation dependent. They can be terminated by a client RTI-PM upon receipt of either a release context request from the client or a shutdown request from the server. They can also be terminated by the server (by indicating deferred end dialogue) during response processing, as long as there are no active context handles at the end of the call.

Table 5-1 Dialogues

Context Type	Context States Allowed	OSI-TP Functional Units Used	Dialogue States Allowed	Type of RPCs Supported
Transaction Enabled	Transactional Non-Transactional	Dialogue Shared Control Commit Unchained	Transactional Non-Transactional	All
Not Transaction Enabled	Non-Transactional	Dialogue Shared Control	Non-Transactional	Non-Transactional including **transaction_optional** without a current transaction

5.3.6 OSI TP Profiles

The TxRPC-ASE refers to OSI TP profiles ATP12 and ATP22 (see ISO/IEC ISP 12061). The following table summarises the OSI TP functional units required by each one of these profiles (a • symbol indicates that the specified functional unit is required for that profile):

Table 5-2 Required OSI TP Functional Units

Functional Units	ATP12	ATP22
Dialogue	•	•
Polarized Control		
Shared Control	•	•
Commit		•
Unchained transactions		•
Chained transactions		
Handshake	(•)	(•)
Recovery		•

Note: The Handshake functional unit is optional for ATP12 and ATP22. However, TxRPC does not use the Handshake functional unit.

TxRPC uses a protocol set, as described in this specification, based on the ATP12 profile when a call request is issued outside of a global transaction. TxRPC uses another protocol set, also as described in this specification, based on the ATP22 profile when the TxRPC-ASE is used within a global transaction. The choice between these sets is automatically supported by the TxRPC-ASE.

5.3.7 Context Trees, Dialogue Trees and Transaction Trees

A client can establish context with any number of servers for the purpose of sending calls to those servers. By establishing context RTI-SUIs form a *context tree*. In a context tree, each node of the tree is an RTI-SUI that is acting as a client to the *subordinate* RTI-SUIs below it and as a server to the *superior* RTI-SUI above it. If an RTI-SUI has no superior, it is referred to as the *root* RTI-SUI. If an RTI-SU has no subordinates, it is referred to as a *leaf* RTI-SUI. The RTI-SUIs between the root and leaf RTI-SUIs are referred to as *intermediate* RTI-SUIs. An example of a context tree is shown in Figure 5-3.

Context trees are supported by dialogues maintained by the RTI-PMs associated with each RTI-SUI. By establishing dialogues RTI-PMs form a *dialogue tree*. For each context tree, a corresponding dialogue tree is created where each node of the dialogue tree is an RTI-PM.

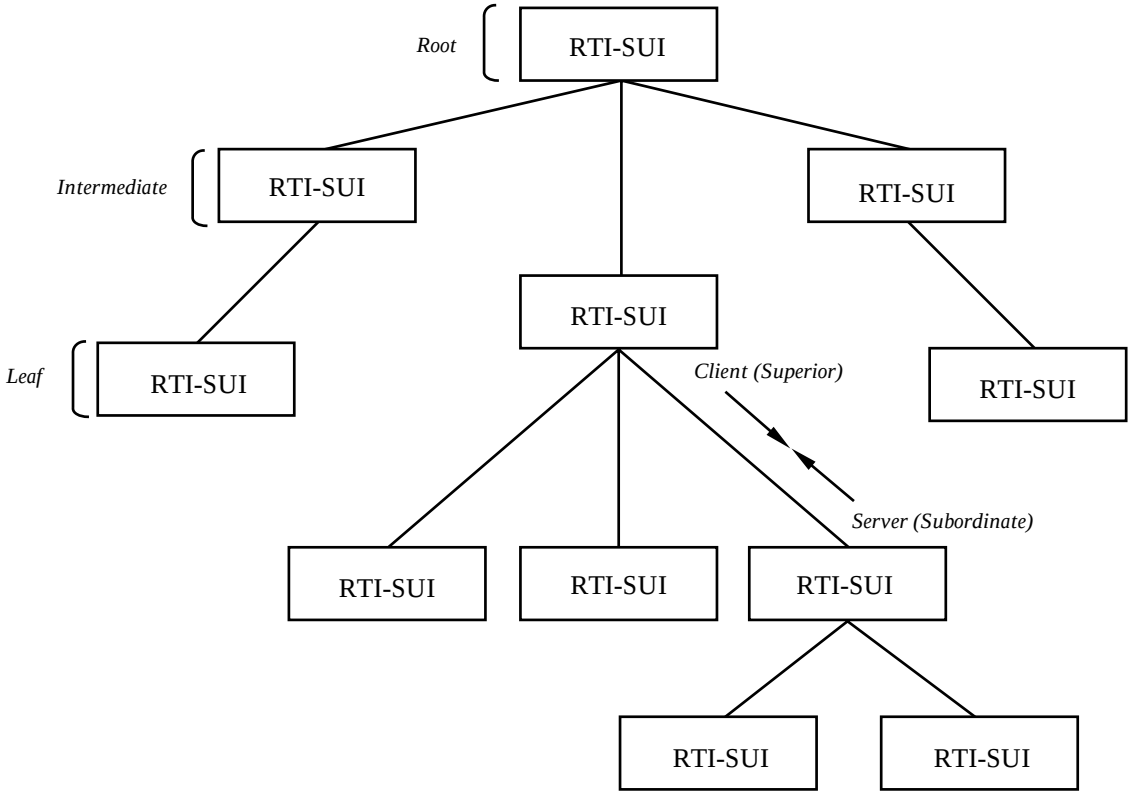


Figure 5-3 RTI-SUI Context Tree

A special case of a context tree is a *transaction tree*. Throughout this document, the term *transaction* refers to a provider supported transaction as defined by OSI TP. A *transaction tree* is a sub-tree of a context tree in which all of the contexts established are transactional contexts. All RTI-SUIs that are part of a transaction tree are referred to as *transaction participants*. All dialogues that support a transaction tree are *transactional dialogues*.

5.3.8 Bound Data

The term *bound data* refers to any persistent data accessed by a transaction participant that exists beyond the duration of a transaction. Bound data is bound by the rules of OSI TP to the state of the transaction for the duration of the transaction.

The state of bound data at the time it is first accessed by a transaction participant is referred to as the *initial state*. The state of bound data immediately after successful completion of a transaction is referred to as the *final state*. Modifications made by the operations of transaction participants change the bound data from the initial state to the final state. The modifications are indivisible and either all are effected (placing the bound data in the final state) or none are effected (placing the bound data in the initial state).

An intermediate state, referred to as the *ready state* identifies the state of bound data in which no further modifications are made to the bound data before it is declared to be in the final state.

5.3.9 Using the RTI Communication Model

Figure 5-4 depicts the RTI Communication Model for a simple system consisting of only two RTI-SUIs. The remote procedure calls made by the two RTI-SUIs are transported over a dialogue that runs between the two RTI-PMs supporting the two RTI-SUIs. The dialogue is carried over an association between two RTI-AEIs. The RTI-SUI on the left is the client RTI-SUI that has caused the client RTI-PM on the left to create the dialogue with the server RTI-PM on the right. Any call requests made by the client RTI-SUI to the client RTI-PM are sent to the server RTI-PM. The server RTI-PM then causes the server RTI-SUI to execute the requested operation.

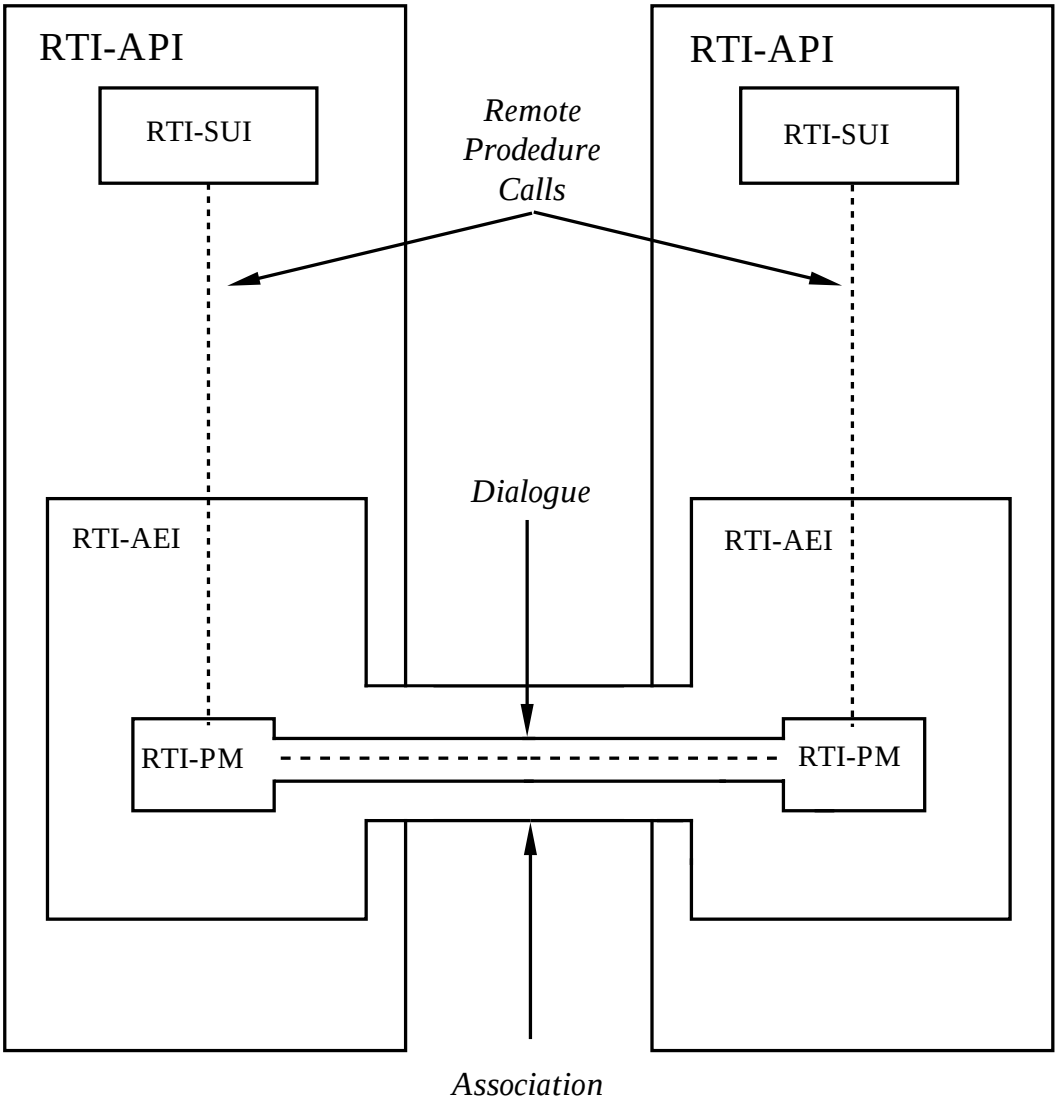


Figure 5-4 RTI Communication Model

5.4 Relationship of the RTI Model to OSI

This section describes the relationship of the RTI model to the OSI TP model and the OSI application layer structure.

The following components are defined or referred to in the OSI TP Model: the Application Process (AP), the Application Process Invocation (API), the Application Entity (AE), the Application Entity Invocation (AEI), the Transaction Processing Service Provider (TPSP), the Transaction Processing Protocol Machine (TPPM), the Transaction Processing Service User (TPSU), the Transaction Processing Service User Invocation (TPSUI).

In the RTI Model these components are mapped as follows:

- AP → RTI-AP
- API → RTI-API
- AE → RTI-AE
- AEI → RTI-AEI
- TPSP → Subfunction of RTI-SP
- TPPM → Subfunction of RTI-PM
- TPSU → RTI-SP
- TPSUI → RTI-PM.

5.5 RTI Naming Model

This section describes the naming information necessary for OSI communication within the RTI Model and shows how those names are mapped to the components of the RTI Model.

5.5.1 OSI Names Used in the RTI Model

The following OSI names are used in the RTI Model:

- application-process-title (AP-Title)
- application-entity-qualifier (AE-Qualifier)
- application-context-name (A-Ctx-Name)
- transaction-process-service-user-title (TPSU-Title)
- abstract-syntax-name (AS-Name).

These names are described in the following sections.

5.5.2 AP-Title

An AP-Title names an AP for the purpose of establishing associations.

A conforming implementation must support an AP-Title of type ObjectID and must be able to receive an AP-Title of type DirectoryName. Optionally, an implementation can support sending an AP-Title of type DirectoryName.

AP-Title in the RTI Model is referred to as an RTI-AP-Title. The scope of an RTI-AP is equivalent to the scope of an AP in OSI TP.

The value of an RTI-AP-Title is implementation-specific.

5.5.3 AE-Qualifier

An AE-Qualifier selects the AE within a particular AP that can accept the association that is being established. The initiating and accepting AEIs are the end points of the association being established.

The AE-Qualifier must be of the same form as the AP-Title. That is, if the AP-Title is of type ObjectID the AE-Qualifier must be of type INTEGER. If the AP-Title is of type DirectoryName the AE-Qualifier must be of type DirectoryName.

5.5.4 A-Ctx-Name

An A-Ctx-Name identifies the application context, and thereby the rules for communication between the two AEIs using the association.

An A-Ctx-Name is of type ObjectID.

All ObjectIDs must be registered with a registration authority. It is the registration authority that fixes the value of the ObjectID. That fixed value is then incorporated into each implementation.

The allowable A-Ctx-Names for the RTI Model are defined in Chapter 6.

5.5.5 TPSU-Title

A TPSU-Title selects the TPSUI within a particular API that can accept the dialogue being established. The initiating and accepting TPSUIs are the end points of the dialogue being established.

TPSU-Title can be either of type Integer or PrintableString.

TPSU-Title is of type PrintableString in the RTI Model.

The allowable TPSU-Titles for the RTI Model are defined in Section 8.3.5 on page 91.

5.5.6 AS-Name

An AS-Name identifies a particular Abstract-Syntax definition.

An AS-Name is of type ObjectID.

Two AS-Names are defined by the RTI Model, one for the DC-ASE abstract syntax, one for the RPC-ASE abstract syntax.

Chapter 6

RTI Application Context Definition

This chapter contains the formal definition of the RTI Application Context. The RTI Model is described in Chapter 5 and the RTI Communication Model in Section 5.3 on page 32.

ISO/IEC 9545 describes an Application Context as ‘‘a set of rules shared in common by two AEIs in order to enable their cooperative operation’’.

The RTI Application Context makes use of the facilities of OSI TP. Note that OSI TP does not define an application context of its own. OSI TP is a set of application layer services and an application layer protocol used by OSI applications that require transaction semantics. Many applications may make use of OSI TP, each defining its own Application Context.

The purpose of an Application Context definition is to restrict options, and apply rules beyond those specified in the standards defining the ASEs that constitute the Application Context.

6.1 Application Context Name

Two application-context names are defined for RTI. They are:

- RTI Application Context with Transactions Enabled

```
{iso(1) national-member-body(2) bsi(826) disc(0) xopen(1050)
 txrpc(6) application-context(1) transactional-enabled(1)}
```

- RTI Application Context without Transactions Enabled

```
{iso(1) national-member-body(2) bsi(826) disc(0) xopen(1050)
 txrpc(6) application-context(1) not-transaction-enabled(2)}
```

6.2 Component ASEs

The component ASEs of the RTI Application Context with Transactions Enabled and the RTI Application Context without Transactions Enabled are identified as part of the RTI Model presented in Chapter 5.

The RTI Application Context without Transactions Enabled includes all ASEs described in Chapter 5, except the CCR-ASE.

The RTI Application Context with Transactions Enabled contains all ASEs described in Chapter 5.

6.3 Application Services

The RTI Application Context with Transactions Enabled and the RTI Application Context without Transactions Enabled do not require the use of any AEI application services other than those defined in this document and in the referenced standards.

6.4 Persistent Application Functions

The RTI Application Context without Transactions Enabled has no persistent application-context rules.

The only persistent application-context rules for the RTI Application Context with Transactions Enabled are those that pertain to bound-data as defined in the OSI TP Model, Service and Protocol standards and described in Section 5.3.8 on page 37.

6.5 SACF Rules

All SACF rules defined for the RTI Application Context with Transactions Enabled or the RTI Application Context without Transactions Enabled that are in addition to, and supplement, those rules that are defined in the referenced standards can be found in Section 8.6 on page 122.

6.6 MACF Rules

All MACF rules defined for the RTI Application Context with Transactions Enabled and the RTI Application Context without Transactions Enabled that are in addition to, and supplement, those rules that are defined in the referenced standards can be found in Section 8.5 on page 112 and Section 8.7 on page 123.

6.7 Optional Features

The RTI Application Context without Transactions Enabled selects the RTI Kernel functional unit and the RTI Non-Transactional functional unit. There are no optional features defined for the RTI Application Context without Transactions Enabled.

The RTI Application Context with Transactions Enabled selects the RTI Kernel functional unit, the RTI Non-Transactional functional unit and the RTI Transactional functional unit.

Rules and restrictions on the selection of RTI functional units (and component ASEs) are defined in Section 7.2 on page 50.

Rules and restrictions on the selection of OSI TP functional units (and component ASEs) are defined in **Functional-Units** on page 93.

6.8 Error Handling

If the rules and constraints of either the RTI Application Context with Transactions Enabled or the RTI Application Context without Transactions Enabled are violated the underlying association is aborted.

6.9 Context Manipulation

There are no Application Context rules that may be dynamically added, removed or modified for either the RTI Application Context with Transactions Enabled or the RTI Application Context without Transactions Enabled.

6.10 Conformance

There are no conformance requirements for the RTI Application Context with Transactions Enabled or the RTI Application Context without Transactions Enabled other than those specified in the referenced standards.

Chapter 7

RTI Service Definition

This chapter contains the definitions of the RTI service primitives. These service primitives, in turn, define the RTI Service. The RTI service primitives exist to enable a client and server to exchange information in a predictable manner.

The definition of the RTI service is presented as follows:

- a summary of the service primitives
- the classification of the service primitives into functional units, according to their ability to be used independently of other functional units
- a description of each functional unit
- a description of the service primitives contained within each functional unit including a request and indication parameter list.

All sequencing rules are implicit in the state tables shown in Section 7.7 on page 75.

7.1 Service Conventions

The service description conventions used in this specification are as close as possible to OSI service description conventions. Where it has been necessary to augment OSI conventions, it has been done so in a way that preserves the general OSI convention style.

In the OSI application layer structure a service primitive is an event that occurs at the boundary between a service user and a service provider in an open system. Similarly, RTI service primitives are modelled as events that occur at the RTI Service Boundary between an RTI-SU (an RTI-SUI) and an RTI-SP (an RTI-PM).

The RTI Service has two *service primitive classes*: requests (req) and indications (ind). Requests are issued by an RTI-SUI and received by an RTI-PM. Indications are issued by an RTI-PM and received by an RTI-SUI. Typically, although not necessarily, requests issued by one RTI-SUI are received as indications by another RTI-SUI. In certain situations requests issued to an RTI-PM are purely local events and do not directly generate protocol flows. Similarly, some indications issued by an RTI-PM are the result of multiple protocol flows or local events (such as communication failures). This sequence of RTI service primitives is shown in Figure 7-1.

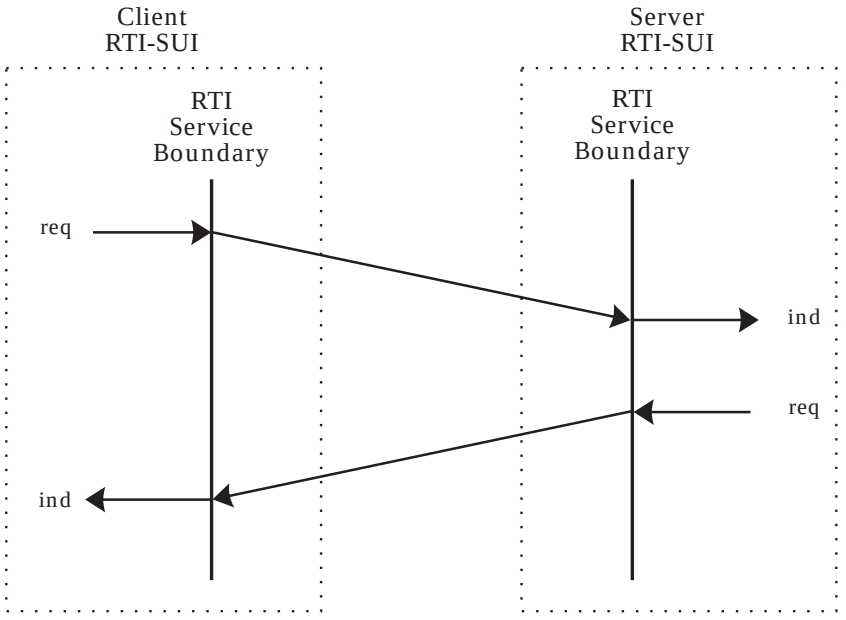


Figure 7-1 RTI Service Primitives Sequencing

Tables are used to describe the component parameters of the RTI service primitives. Each table consists of up to three columns, containing the name of the service parameter, the request parameter usage, and the indication parameter usage. Each parameter is listed on a separate line. Under the appropriate service primitive class columns, a code specifies the usage of the parameter for the service primitive class specified in the vertical column:

M Parameter is mandatory.

U Parameter is a user option, and may or need not be provided depending on the dynamic requirements of the RTI client or RTI server.

O Parameter is an RTI-PM option, and may or need not be provided depending on the dynamic requirements of the provider.

C Parameter is conditional upon other parameters or the environment of a client or server.

(blank) Parameter is never present.

The code **(=)** following one of the codes **M**, **U**, **O** or **C** indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table. (For instance, an **M(=)** code in the indication service primitive column and an **M** in the request service primitive column means that the parameter in the indication primitive column is semantically equivalent to that in the request primitive column.)

The descriptions of the parameters specify the allowable data values.

7.2 Service Functional Unit Description

The following functional units are defined:

- **Kernel**

  The Kernel functional unit provides the RTI service primitives for establishing context and calling remote tasks. The Kernel function unit is used in conjunction with the Non-Transactional functional unit and, optionally, the Transactional functional unit.

- **Non-Transactional**

  The Non-Transactional (Non-Trans) functional unit augments the Kernel functional unit. It provides the necessary additional context release primitive for use with the Kernel service primitives when non-transactional dialogues are terminated.

- **Transactional**

  The Transactional (Trans) functional unit augments the Kernel and Non-Transactional functional units. It provides additional transaction semantics service primitives when calls with transaction semantics are required.

Functional units are selected by an RTI-PM when context is established.

The Kernel and Non-Transactional functional units are always selected. Additionally, the Transactional functional unit may also be selected.

7.3 Summary of Service Primitives

The RTI service is invoked using a sequence of RTI service primitives. Table 7-1 lists:

- the RTI service primitives
- the service primitive type (req or ind) available to clients and servers
- the functional unit to which the service primitive belongs (Kernel, Non-Trans or Trans)
- the section of this document that fully describes the service primitive.

Table 7-1 RTI Service Primitives

Service Name	Client Primitives	Server Primitives	Functional Unit	See
RTI-ESTABLISH-CONTEXT	req		Kernel	Section 7.4.1 on page 53.
RTI-CALL-TASK	req	ind	Kernel	Section 7.4.2 on page 56.
RTI-CANCEL-CALL	req	ind	Kernel	Section 7.4.3 on page 59.
RTI-CALL-FAILURE	ind		Kernel	Section 7.4.4 on page 60.
RTI-CALL-RESULT	ind	req	Kernel	Section 7.4.5 on page 63.
RTI-RELEASE-CONTEXT	req/ind		Non-Trans	Section 7.5.1 on page 65.
RTI-HEURISTIC-REPORT	ind		Trans	Section 7.6.1 on page 67
RTI-ROLLBACK-TRANS	req/ind	req/ind	Trans	Section 7.6.2 on page 68.
RTI-END-TRANS	req		Trans	Section 7.6.3 on page 69.
RTI-PREPARE-TRANS		ind	Trans	Section 7.6.4 on page 70.
RTI-TRANS-READY		req/ind	Trans	Section 7.6.5 on page 71.
RTI-COMMIT-TRANS	req/ind	ind	Trans	Section 7.6.6 on page 72.
RTI-TRANS-DONE	req	req	Trans	Section 7.6.7 on page 73.
RTI-TRANS-COMPLETE	ind	ind	Trans	Section 7.6.8 on page 74.

7.4 Kernel Functional Unit

The Kernel functional unit provides the RTI service primitives for establishing context and calling remote tasks. The Kernel functional unit is used in conjunction with the Non-Transactional and, optionally, the Transactional functional units.

These primitives enable an RTI-SUI to:

- establish client-server context
- request a call
- cancel an outstanding call
- receive the result of a call
- receive indication of a failure of a call.

The Kernel functional unit is always selected. Both client and server RTI-PMs must support this functional unit. The functional units that are used by a particular context are determined when the context is established. Rules that describe how the functional units may be combined are described in Section 7.2 on page 50.

7.4.1 RTI-ESTABLISH-CONTEXT request

Function

An RTI-ESTABLISH-CONTEXT request is issued by a client to an RTI-PM to request that a context be established between a client and a server.

This service primitive relates to one particular context.

Parameters

Table 7-2 RTI-ESTABLISH-CONTEXT Parameters

Parameter Name	Req
RTI-AP-Title	M
RTI-AE-Qualifier	U
Object-UUID	U
Client-Name	M
Client-Authenticator-Type	M
Client-Authenticator	M
Interface-UUID	M
Interface-Version-Major	M
Interface-Version-Minor	M
Context-Type	M

RTI-AP-Title
: This parameter is supplied by the client. It specifies the name of the server RTI-API that processes calls made by the client within the context being established.

  This parameter can have any value that is a valid RTI-AP-Title (AP-Title of an AP that supports the RTI Service).

RTI-AE-Qualifier
: This parameter is supplied by the client. It specifies the AE-Qualifier in which this server RTI-API resides. This parameter is optional.

  This parameter can have any value corresponding to a valid RTI-AE-Qualifier.

Object-UUID
: This parameter is supplied by the client. It specifies an object UUID that can be used to affect the selection of the server and the manager function within the server. This parameter is optional.

  This parameter can have any value that is a valid UUID as defined in the referenced X/Open **DCE RPC** specification.

Client-Name
: This parameter is supplied by the client. It specifies the name of the client issuing the request. This parameter is used in conjunction with the Client-Authenticator to verify the authenticity of the client.

  This parameter can have any value that is a valid VisibleString.

Client-Authenticator-Type
: This parameter is supplied by the client. It specifies the type of the Client-Authenticator

parameter.

This parameter can have one of the following values:

DEFAULT-SECURITY

CUSTOMER-WRITTEN-SECURITY

Client-Authenticator
This parameter is supplied by the client. This parameter is used in conjunction with the Client-Name to verify the authenticity of the client in an implementation-dependent manner.

If the Client-Authenticator-Type is DEFAULT-SECURITY, this parameter can have any value that is a valid VisibleString.

If the Client-Authenticator-Type is CUSTOMER-WRITTEN-SECURITY, this parameter can have any value that is a valid OCTET STRING.

Interface-UUID
This parameter is supplied by the client. It specifies the UUID of the server interface to use for calls to be made by the client within the context being established.

This parameter can have any value that is a valid UUID as defined in the referenced X/Open **DCE RPC** specification.

Interface-Version-Major
This parameter is supplied by the client. It specifies the major version number of the server interface identified by Interface-UUID.

This parameter can have any value that is a 16-bit non-negative integer.

Interface-Version-Minor
This parameter is supplied by the client. It specifies the minor version number of the server interface identified by Interface-UUID.

This parameter can have any value that is a 16-bit non-negative integer.

Context-Type
This parameter is supplied by the client. It specifies whether the context being established is enabled for transactional RPCs.

This parameter can have one of the following values:

TRANSACTION-ENABLED
This context is established with transactions enabled; both transactional and non-transactional RPCs are supported.

NOT-TRANSACTION-ENABLED
This context is established without transactions enabled; only non-transactional RPCs are supported.

Usage

- Only a client can issue an RTI-ESTABLISH-CONTEXT request.
- No other RTI service primitives can be issued by a client without first establishing context with a server. Therefore an RTI-ESTABLISH-CONTEXT request must be issued by a client prior to issuing any other RTI service primitive.
- An RTI-ESTABLISH-CONTEXT request does not result in an immediate flow of protocol; it sets up a relationship between a client and a server and specifies the characteristics of that relationship.
- Context is said to exist at a client from the moment an RTI-ESTABLISH-CONTEXT request is issued until some other service primitive releases that context.
- The use of transactional RPCs requires that the context be established with Context-Type set to TRANSACTION-ENABLED. If only non-transactional RPCs are required, NOT-TRANSACTION-ENABLED can be specified. The control of this option is implementation defined.
- Once an RTI-ESTABLISH-CONTEXT request has been issued by a client, only one of the following events can occur:
  - — issue an RTI-RELEASE-CONTEXT request
  - — issue an RTI-CALL-TASK request
  - — issue an RTI-RELEASE-CONTEXT indication.

7.4.2 RTI-CALL-TASK request and indication

Function

An RTI-CALL-TASK request is issued by a client to an RTI-PM to request a call.

An RTI-CALL-TASK indication is issued by an RTI-PM to a server to initiate a call.

These service primitives relate to one particular context.

Parameters

Table 7-3 RTI-CALL-TASK Parameters

Parameter Name	Req	Ind
Client-Name		M
Client-Authenticator-Type		M
Client-Authenticator		M
Context-Type		M
Interface-UUID	M	M(=)
Interface-Version-Major	M	M(=)
Interface-Version-Minor	M	M(=)
Object-UUID		O(=)
Transaction-Attribute	M	M(=)
Operation-Number	M	M(=)
Arguments	M	M(=)

Client-Name
: This parameter is supplied by the server RTI-PM. The value of this parameter depends upon the context in which the call was made. It specifies the name of the client issuing the request. This parameter is used in conjunction with the Client-Authenticator to verify the authenticity of the client.

  This parameter can have any value that is a valid VisibleString.

Client-Authenticator-Type
: This parameter is supplied by the server RTI-PM. The value of this parameter depends upon the context in which the call was made. It specifies the type of the Client-Authenticator parameter.

  This parameter can have one of the following values:

  DEFAULT-SECURITY
  : The context being established uses default security.

  CUSTOMER-WRITTEN-SECURITY
  : The context being established uses customer written security.

Client-Authenticator
: This parameter is supplied by the server RTI-PM. The value of this parameter depends upon the context in which the call was made. This parameter is used in conjunction with the Client-Name to verify the authenticity of the client.

  If the Client-Authenticator-Type is DEFAULT-SECURITY, this parameter can have any value that is a valid VisibleString.

If the Client-Authenticator-Type is CUSTOMER-WRITTEN-SECURITY, this parameter can have any value that is a valid OCTET STRING.

Context-Type
This parameter is supplied by the server RTI-PM. It specifies whether the call is being made within a context with transactions enabled.

This parameter can have one of the following values:

TRANSACTION-ENABLED
This context was established with transactions enabled; both transactional and non-transactional RPCs are supported.

NOT-TRANSACTION-ENABLED
This context was established without transactions enabled; only non-transactional RPCs are supported.

Interface-UUID
This parameter is supplied by the client. It specifies the UUID of the server interface to use for this call.

This parameter can have any value that is a valid UUID as defined in the referenced X/Open **DCE RPC** specification.

Interface-Version-Major
This parameter is supplied by the client. It specifies the major version number of the server interface identified by Interface-UUID.

This parameter can have any value that is a 16-bit non-negative integer.

Interface-Version-Minor
This parameter is supplied by the client. It specifies the minor version number of the server interface identified by Interface-UUID.

This parameter can have any value that is a 16-bit non-negative integer.

Object-UUID
This parameter is supplied by the server RTI-PM. It is the object UUID specified when the RTI context was established.

This parameter can have any value that is a valid UUID as defined in the referenced X/Open **DCE RPC** specification.

Transaction-Attribute
This parameter is supplied by the client. It specifies the transaction attribute defined for the operation.

This parameter can have one of the following values:

TRANSACTION-MANDATORY
The requested operation has the IDL **transaction_mandatory** attribute.

TRANSACTION-OPTIONAL
The requested operation has the IDL **transaction_optional** attribute.

TRANSACTION-NONE
The requested operation has neither the IDL **transaction_mandatory** nor **transaction_optional** attribute.

Operation-Number
This parameter is supplied by the client. It specifies the operation number of the call within

the interface specified by the Interface-UUID, which is associated with the context in which the call was made.

This parameter can have any value that is a 16-bit non-negative integer as defined in **Parameters** on page 96.

Arguments

This parameter is supplied by the client. It contains the input parameters to the call being made.

Usage

- An RTI-CALL-TASK request can only be issued within an established context.
- Receipt of an RTI-CALL-TASK indication completes the context establishment with a server.
- An operation with a Transaction-Attribute of TRANSACTION-MANDATORY can only be issued within the scope of a transaction. The Context-Type is TRANSACTION-ENABLED.
- If an operation with a Transaction-Attribute of TRANSACTION-MANDATORY or TRANSACTION-OPTIONAL is issued within the scope of a transaction, the operation is a transactional RPC. If the context is not already transactional, it is made transactional and the associated dialogue is included in the global transaction. The manager function executes under the global transaction. The Context-Type is TRANSACTION-ENABLED.
- If an operation with a Transaction-Attribute of TRANSACTION-OPTIONAL is issued outside the scope of a transaction, the manager function begins execution in non-transaction mode. Note that in this case, the context and the associated dialogue are necessarily non-transactional. The Context-Type may be either TRANSACTION-ENABLED or NOT-TRANSACTION-ENABLED.
- If the Transaction-Attribute is TRANSACTION-NONE the manager function begins execution in non-transaction mode. This is true even if the call is issued within the scope of a global transaction and the context is transactional and the associated dialogue has been included in the transaction. The Context-Type may be either TRANSACTION-ENABLED or NOT-TRANSACTION-ENABLED.
- Once an RTI-CALL-TASK request has been issued by a client, only one of the following events can occur:
  - — issue an RTI-CANCEL-CALL request
  - — receive an RTI-CALL-RESULT indication
  - — receive an RTI-CALL-FAILURE indication
  - — receive an RTI-ROLLBACK-TRANS indication.
- Once an RTI-CALL-TASK indication has been received by a server, only one of the following events can occur:
  - — issue an RTI-CALL-RESULT request
  - — receive an RTI-CANCEL-CALL indication
  - — receive an RTI-ROLLBACK-TRANS indication.

7.4.3 RTI-CANCEL-CALL request and indication

Function

An RTI-CANCEL-CALL request is issued by a client to an RTI-PM to request cancellation of an outstanding call.

An RTI-CANCEL-CALL indication is issued by an RTI-PM to a server to indicate that an outstanding call is requested to be cancelled.

These service primitives relate to one context.

Parameters

None.

Usage

- An RTI-CANCEL-CALL request can only be issued within an established context.
- An RTI-CANCEL-CALL request can only be issued by a client after an RTI-CALL-TASK request has been issued and before an RTI-CALL-RESULT or RTI-CALL-FAILURE indication is received. That is, while an outstanding call exists.
- Multiple RTI-CANCEL-CALL requests can be made by a client.
- If multiple RTI-CANCEL-CALL requests are made by a client, multiple RTI-CANCEL-CALL indications may be received by the server processing the call.
- RTI-CANCEL-CALL indications are received by a server only after an RTI-CALL-TASK indication has been received and before an RTI-CALL-RESULT request is issued. That is, while an outstanding call exists.
- The context in which an RTI-CANCEL-CALL request is issued identifies the server that has initiated the call that is to be cancelled.
- Once an RTI-CANCEL-CALL request has been issued by a client, only one of the following events can occur:
  - — receive an RTI-CALL-RESULT indication
  - — receive an RTI-CALL-FAILURE indication
  - — receive an RTI-ROLLBACK-TRANS indication.
- Once an RTI-CANCEL-CALL indication has been received by a server, only one of the following events can occur:
  - — issue an RTI-CALL-RESULT request
  - — receive an RTI-CANCEL-CALL indication
  - — receive an RTI-ROLLBACK-TRANS indication.

7.4.4 RTI-CALL-FAILURE indication

Function

An RTI-CALL-FAILURE indication is issued by an RTI-PM to indicate to a client that a call has failed.

This indication relates to one particular context.

Parameters

Table 7-4 RTI-CALL-FAILURE Parameters

Parameter Name	Ind
Reason	M

Reason
This parameter is supplied by the RTI-PM. It specifies the reason the call has failed.

This parameter can have one of the following values:

RTI-SERVICE-UNKNOWN
The server RTI-API specified on the RTI-ESTABLISH-CONTEXT request does not support the RTI Service.

The specified context is no longer valid.

PROTOCOL-VERSION-NOT-SUPPORTED
The server RTI-API specified on the RTI-ESTABLISH-CONTEXT request does not support the version of RTI Service specified by the client RTI-API.

The specified context is no longer valid.

CONTEXT-TYPE-NOT-SUPPORTED
The server RTI-API specified on the RTI-ESTABLISH-CONTEXT request does not support the specified context type.

The specified context is no longer valid.

TRANSACTION-ATTRIBUTE-MISMATCH
The Transaction-Attribute specified in the RTI-CALL-TASK request does not match the Transaction-Attribute known in the server.

PERMANENT-COMMUNICATION-FAILURE
A permanent communication failure between the client and server RTI-APIs has been detected. Re-establishing the context will most likely be unsuccessful.

The specified context is no longer valid.

TRANSIENT-COMMUNICATION-FAILURE
A transient communication failure between the client and server RTI-APIs has been detected. Re-establishing the context is possible.

PROTOCOL-MACHINE-FAILURE
A fatal RTI-PM failure has been detected.

The specified context is no longer valid.

INTERFACE-UNKNOWN
: The server RTI-API specified on the RTI-ESTABLISH-CONTEXT request does not support the Interface specified on the RTI-CALL-TASK request.

INTERFACE-PERMANENTLY-UNAVAILABLE
: The server RTI-API specified on the RTI-ESTABLISH-CONTEXT request supports the Interface specified on the RTI-CALL-TASK request but does not support the Interface Version specified on the RTI-CALL-TASK request.

INTERFACE-TEMPORARILY-UNAVAILABLE
: The server RTI-API specified on the RTI-ESTABLISH-CONTEXT request supports the requested Interface and Interface Version specified on the RTI-CALL-TASK request but does not have sufficient resources at this time to allow access.

ROLLBACK-IN-PROGRESS
: A rollback of the current transaction has been detected.

  The specified context is no longer valid (if there are no active context handles).

REASON-NOT-SPECIFIED
: Reason not specified.

  The specified context is no longer valid.

RPC-ACCESS-VIOLATION
: The call has resulted in an access violation.

RPC-CANCEL
: The call has been cancelled.

RPC-FLOATING-DIVIDE-BY-ZERO
: The call has resulted in a floating-point divide by zero.

RPC-FLOATING-ERROR
: The call has resulted in a floating-point error.

RPC-FLOATING-OVERFLOW
: The call has resulted in a floating-point overflow.

RPC-FLOATING-UNDERFLOW
: The call has resulted in a floating-point underflow.

RPC-INSUFFICIENT-RESOURCES
: The server does not have sufficient resources to process the call.

RPC-INTEGER-DIVIDE-BY-ZERO
: The call has resulted in an integer divide by zero.

RPC-INTEGER-OVERFLOW
: The call has resulted in an integer overflow.

RPC-INVALID-OPERATION-NUMBER
: An invalid operation number was specified on the RTI-CALL-TASK request.

RPC-INVOCATION-FAILURE
: A fatal error was encountered while attempting to invoke the call.

RPC-MARSHALLING-ERROR
: A marshalling error has been encountered.

RPC-PROTOCOL-ERROR
: An RPC protocol error has been encountered.

RPC-REASON-NOT-SPECIFIED
Reason not specified.

Usage

- Once an RTI-CALL-FAILURE indication has been received by a client, only one of the following events can occur:
  - — issue an RTI-RELEASE-CONTEXT request
  - — issue an RTI-CALL-TASK request
  - — issue an RTI-END-TRANS request
  - — issue an RTI-ROLLBACK-TRANS request
  - — receive an RTI-RELEASE-CONTEXT indication
  - — receive an RTI-ROLLBACK-TRANS indication.

7.4.5 RTI-CALL-RESULT request and indication

Function

An RTI-CALL-RESULT request is issued by a server to an RTI-PM to return output parameters after the completion of a call.

An RTI-CALL-RESULT indication is issued by an RTI-PM to a client to return output parameters upon the completion of a call.

These service primitives relate to one particular context.

Parameters

Table 7-5 RTI-CALL-RESULT Parameters

Parameter Name	Req	Ind
Arguments	M	M(=)

Arguments
This parameter is supplied by the server. It contains the output parameters to the call being completed.

Usage

- These service primitives can only be issued and received within an established context.
- An RTI-CALL-RESULT request can only be issued by a server for the purpose of reporting back the result of a call.
- The context in which an RTI-CALL-RESULT request is issued identifies the client that requested the call and to which the results are to be returned.
- Once an RTI-CALL-RESULT request has been issued by a server, only one of the following events can occur:
  - — issue an RTI-ROLLBACK-TRANS request
  - — receive an RTI-CALL-TASK indication
  - — receive an RTI-ROLLBACK-TRANS indication
  - — receive an RTI-PREPARE-TRANS indication.
- Once an RTI-CALL-RESULT indication has been received by a client, only one of the following events can occur:
  - — issue an RTI-RELEASE-CONTEXT request
  - — issue an RTI-CALL-TASK request
  - — issue an RTI-END-TRANS request
  - — issue an RTI-ROLLBACK-TRANS request
  - — receive an RTI-RELEASE-CONTEXT indication
  - — receive an RTI-ROLLBACK-TRANS indication.

7.5 Non-Transactional Functional Unit

The Non-Transactional functional unit augments the Kernel functional unit. It provides the necessary context release primitive for use with the Kernel functional unit service primitives when non-transactional contexts are terminated.

The Non-Transactional functional unit is always selected. Both client and server RTI-PMs must support this functional unit. The functional units that are used by a particular context are determined when the context is established. Rules that describe how the functional units may be combined are described in Section 7.2 on page 50.

7.5.1 RTI-RELEASE-CONTEXT request and indication

Function

An RTI-RELEASE-CONTEXT request is issued by a client to an RTI-PM to request the release of an established context.

An RTI-RELEASE-CONTEXT indication is issued by an RTI-PM to a client to indicate that an established context has been released.

These service primitives relate to one particular context.

Parameters

None.

Usage

- If an RTI-RELEASE-CONTEXT request is issued or an RTI-RELEASE-CONTEXT indication received, no further requests can be issued or indications received in context of the released context. The context is released.
- All service primitives issued within a context by a client, prior to the release of that context, are guaranteed to be delivered to the corresponding server before the context is released.
- An RTI-RELEASE-CONTEXT request cannot be issued within the context of an outstanding call.

7.6 Transactional Functional Unit

The Transactional functional unit augments the Kernel functional unit. It provides transaction service primitives for use with the Kernel and Non-Transactional functional unit service primitives. These primitives enable the RTI-SUI to:

- initiate transaction termination
- rollback (undo) a transaction
- participate in two-phase commitment of the transaction
- receive the results of the two-phase commitment process.

The Transactional functional unit is selected for contexts with transactions enabled. Both client and server RTI-PMs must support this functional unit. The functional units that are used by a particular context are determined when the context is established. Rules that describe how the functional units may be combined are described in Section 7.2 on page 50.

7.6.1 RTI-HEURISTIC-REPORT indication

Function

An RTI-HEURISTIC-REPORT indication is issued by an RTI-PM to indicate to a client that an exception condition has been detected within the transaction tree.

This indication reports only non-fatal exception conditions, the established context is not released by this indication.

Heuristic reports do not require any action on the part of a client.

This indication relates to only one context.

Parameters

Table 7-6 RTI-HEURISTIC-REPORT Parameters

Parameter Name	Ind
Diagnostic	M

Diagnostic
This parameter is supplied by the RTI-PM. It specifies the exception that occurred within the transaction tree.

This parameter can have one of the following values:

HEURISTIC-MIX
The bound data handled by the RTI-SUI are in a state that is inconsistent with the outcome of the transaction.

HEURISTIC-HAZARD
A communication failure has occurred within the RTI-SUI that may prevent reporting of data inconsistency.

Usage

- Only a client can receive this indication.
- This indication is issued by an RTI-PM when a heuristic decision made by a transaction participant in the transaction tree below a client causes an exception condition.
- The established context is not terminated by this indication.

7.6.2 RTI-ROLLBACK-TRANS request and indication

Function

An RTI-ROLLBACK-TRANS request is issued by an RTI-SUI to an RTI-PM to initiate the rollback of a transaction.

An RTI-ROLLBACK-TRANS indication is issued by an RTI-PM to an RTI-SUI to indicate that a transaction is to be rolled back.

These service primitives relate to all transactional contexts.

Parameters

None.

Usage

- A server cannot issue an RTI-ROLLBACK-TRANS request within the context of an outstanding call. If a server wants to request a rollback within the context of an outstanding call, an RTI-CALL-RESULT request must be issued with appropriate arguments and wait for a resulting RTI-ROLLBACK-TRANS indication. This rule exists because the TP-ROLLBACK service does not allow rollback reason codes. By using the call result service primitives to request a rollback, a server is able to supply a rollback reason to a client.
- If an RTI-ROLLBACK-TRANS request is issued by an RTI-SUI, all RTI-SUIs in that RTI-SUI's transaction tree receive RTI-ROLLBACK-TRANS indications.
- If for any reason an RTI-PM detects a failure that prevents a transaction from committing, it issues an RTI-ROLLBACK-TRANS indication to all RTI-SUIs in the transaction tree.
- Once an RTI-ROLLBACK-TRANS request is issued or an RTI-ROLLBACK-TRANS indication is received no Kernel functional unit service primitives may be issued or received for contexts that do not have any active context handles.
- Upon receipt of an RTI-ROLLBACK-TRANS indication, an RTI-SUI must place all recoverable resources in their initial state. When this has been completed an RTI-TRANS-DONE request must be issued.
- An RTI-ROLLBACK-TRANS request cannot be issued during a transaction by an RTI-SUI if that RTI-SUI has already issued an RTI-TRANS-READY request.
- All transactional contexts that do not have any active context handles associated with them are released at the end of a transaction (in this case rollback completion — signalled by the receipt of an RTI-TRANS-COMPLETE indication).

7.6.3 RTI-END-TRANS request

Function

An RTI-END-TRANS request is issued by a client that is the root of a transaction tree to an RTI-PM to initiate the termination of a transaction.

This request signals the beginning of phase one of the two-phase commitment process. It indicates that a client issues no further requests and requests that all transaction participants issue RTI-TRANS-READY requests.

If all transaction participants in the transaction tree agree to commit and no service-provider condition prevents commitment, a transaction commits.

If at least one transaction participant in the transaction tree refuses to commit or if a service-provider condition prevents commitment, the transaction is rolled back.

This service primitive relates to all transactional contexts.

Parameters

None.

Usage

- This request can only be issued by a client that is the root of a transaction tree.
- This request can only be issued when both of the following are true:
  - — All processing for the transaction by a client has been successfully completed. That is, all RTI-CALL-TASK requests issued by a client, relating to a transaction, have been completed.
  - — A client issues no further call requests to a server during a transaction.
- The client RTI-PM sets deferred end dialogue for all transactional dialogues that do not have any active context handles associated with them. This results in dialogue termination at transaction completion.
- After this request has been issued, a client must place its bound data into the ready state and then issue an RTI-TRANS-READY request. If for some reason a client cannot place its bound data into the ready state, it must issue an RTI-ROLLBACK-TRANS request and return its bound data to the initial state.
- After an RTI-END-TRANS request is issued, all RTI-SUIs in the transaction tree receive RTI-PREPARE-TRANS indications.

7.6.4 RTI-PREPARE-TRANS indication

Function

An RTI-PREPARE-TRANS indication is issued by an RTI-PM to a server to indicate that a transaction is to be terminated, and that all server recoverable resources must be placed in the ready state.

This indication signals the beginning of phase one of the two-phase commitment process for a server.

This indication relates to all transactional contexts.

Parameters

None.

Usage

- This indication can only be received by a server.
- This indication can only be issued when both of of the following are true:
  - — All processing for a transaction by a client has been successfully completed. That is, all RTI-CALL-TASK requests issued by a client, relating to a transaction, have been completed.
  - — Placement of all server recoverable resources in the ready state is being requested.
- After all server recoverable resources are placed in the ready state, a server can issue an RTI-TRANS-READY request.
- If for some reason a server cannot place its recoverable resources in the ready state, it must issue an RTI-ROLLBACK-TRANS request.

7.6.5 RTI-TRANS-READY request and indication

Function

An RTI-TRANS-READY request is issued by a server RTI-SUI to an RTI-PM to indicate that all RTI-SUI recoverable resources have been placed in the ready state. The request is an RTI-SUI's vote to proceed with commitment of a transaction.

The request signals the end of phase one of the two-phase commitment process for a server.

The request relates to all transactional contexts.

An RTI-TRANS-READY indication is issued by the RTI-PM to a client RTI-SUI to indicate that the transaction subtree has all been placed into the ready state. This indication signals the end of phase I for a client and relates to all transactional contexts.

Parameters

None.

Usage

- The request can only be issued by a server after receiving an RTI-PREPARE-TRANS indication.
- After a server RTI-SUI has issued this request, no other request may be issued until an RTI-COMMIT-TRANS indication or an RTI-ROLLBACK-TRANS indication has been received.
- The indication can only be received after a client RTI-SUI has issued an RTI-END-TRANS request. It must be received before a client RTI-SUI issues an RTI-COMMIT request.
- After an RTI-TRANS-READY indication has been received, the transaction may still be rolled back by a client RTI-SUI by issuing an RTI-ROLLBACK-TRANS request.

7.6.6 RTI-COMMIT-TRANS request and indication

Function

An RTI-COMMIT-TRANS request is issued by a client RTI-SUI to indicate that the outcome of a transaction is commitment, and to signal the beginning of phase two of the two-phase commitment process for this RTI-SUI. The RTI-PM then generates an RTI-COMMIT-TRANS indication to acknowledge the request.

An RTI-COMMIT-TRANS indication is issued by an RTI-PM to an RTI-SUI to indicate that the outcome of a transaction is commitment.

The RTI-COMMIT-TRANS indication signals the beginning of phase two of the two-phase commitment process for a server RTI-SUI.

The RTI-COMMIT-TRANS indication relates to all transactional contexts.

Parameters

None.

Usage

- A client RTI-SUI may issue an RTI-COMMIT-TRANS request only after receiving an RTI-TRANS-READY indication.
- Upon receipt of the RTI-COMMIT-TRANS indication, an RTI-SUI is required to place all its recoverable resources in their final state. When all its recoverable resources are in the final state, an RTI-TRANS-DONE request must be issued.
- After the receipt of an RTI-COMMIT-TRANS indication an RTI-SUI does not receive an RTI-ROLLBACK-TRANS indication for the duration of a transaction.

7.6.7 RTI-TRANS-DONE request

Function

An RTI-TRANS-DONE request is issued by an RTI-SUI to an RTI-PM to indicate that all actions associated with a transaction have been completed, and that all its recoverable resources are in either their initial or final states (rolled back or committed respectively).

If this request is being issued in response to an RTI-COMMIT-TRANS indication, then this service primitive signals the end of phase two of the two-phase commitment process.

If this request is being issued in response to an RTI-ROLLBACK-TRANS indication, then this service primitive signals the end of the rollback process.

This request relates to all transactional contexts.

Parameters

Table 7-7 RTI-TRANS-DONE Parameters

Parameter Name	Req
Heuristic-Report	U

Heuristic-Report
: This parameter is supplied by the server to indicate that a heuristic decision has been made and is to be reported.

  This parameter can have one of the following values:

  HEURISTIC-MIX
  : The bound data handled by the RTI-SUI are in a state that is inconsistent with the outcome of the transaction.

  HEURISTIC-HAZARD
  : A communication failure has occurred within the RTI-SUI that may prevent the reporting of data inconsistency.

Usage

- This request can only be issued by an RTI-SUI to an RTI-PM after one of the following events:
  - — receiving an RTI-COMMIT-TRANS indication
  - — receiving an RTI-ROLLBACK-TRANS indication
  - — issuing an RTI-ROLLBACK-TRANS request.

7.6.8 RTI-TRANS-COMPLETE indication

Function

An RTI-TRANS-COMPLETE indication is issued by an RTI-PM to an RTI-SUI to indicate that all transaction participants have completed the termination of a transaction. Receipt of this indication indicates that all recoverable resources are in either their initial or final states (rolled back or committed respectively).

This indication signals the end of a transaction.

This indication releases all transactional contexts that do not have any active context handles associated with them.

Parameters

None.

Usage

- This indication signals that a transaction has been completed, and that all resources related may be released.
- This indication releases all transactional contexts that do not have any active context handles associated with them.

7.7 Sequencing Rules and State Tables

The RTI Service State Tables describe the allowed sequence of service events for a given RTI Service boundary (between an RTI-SUI and an RTI-PM). The text of the previous chapters takes precedence over these state tables.

A separate state of the tables is maintained for each context established for an RTI-SUI. Service primitives that relate to a single context affect only the state for that particular context. Some service primitives affect all transactional contexts; these service primitives are shown in **bold**. Any service primitives so marked can be issued or received only once for all transactional contexts.

The state tables specify predicates that must be satisfied in order for individual service primitives to be allowed in a given state. These predicates are based on the values of variables.

The state tables also specify actions to be performed. These actions involve setting variables to specified values. The referenced variables are of two types. One type is private to a single context, these are referred to as *local variables*. The other is shared among all contexts established for an RTI-SUI, these are referred to as *global variables*.

The overall state of an RTI-SUI consists of the state of all contexts established for that RTI-SUI, together with the associated values of all local and global variables.

In order to simplify and to enhance readability two state tables are defined, the RTI Service Transactional State Table and the RTI Service Non-Transactional State Table. State transitions for non-transactional contexts are described by the non-transactional state table and state transitions for transactional contexts are described by the transactional state table. Transitions occur between these two state tables as context changes from non-transactional to transactional and vice versa.

The complete state of an RTI-SUI is represented by a collection of one or more of these tables, one for each context established. The RTI Service State Tables are listed in Section 7.7.5 on page 78.

When a new context is created, its initial state is I (the Idle State) in the RTI Service Non-Transactional State Table, and all variables local to the context are set to FALSE.

7.7.1 State Table Conventions

In each state table:

- Each column (except the two left-most columns) represents a state.
- Each row (except the first) represents a service primitive.
- Each cell of the table represents a state transition.

State transitions are controlled by two factors, the incoming event triggering the transition and zero or more predicate variables.

Service primitives that constitute incoming events are listed in the left-most column of the state table (labelled **Event**). When a service primitive is shown in **bold**, the event is applied to all transactional contexts. The RTI Service Primitives are discussed in Section 7.3 on page 51 and summarised in Table 7-1 on page 51.

Variables are of the form *Vx*, where *x* is the two letter abbreviated name of the variable. If a variable is listed in the state table prefixed by ¬ (logical NOT) then the value of that variable must be FALSE in order for the transition predicated by that variable to occur. If a variable has no prefix, the value of that variable must be TRUE in order for the transition predicated by that variable to occur. The RTI Service variables are listed in Section 7.7.2 on page 76.

Variables may be listed as a precondition predicating all transitions associated with a particular service primitive or may be listed in a particular cell of the state table predicating only the transition represented by that cell. Precondition variables are listed in the second column of the table (labelled **Pre.**). All precondition variables must evaluate as specified in order for any transition in that row of the state table to occur. Cell variables (when they appear) are listed first in each cell. All cell variables must evaluate as specified in order for the transition represented by that cell to occur.

In addition to cell variables each cell of the state table may contain a set of actions to be performed before the transition represented by the cell is made. Actions are of the form [*x*] where *x* is the number of the action to be performed. The RTI Service Actions are listed in Section 7.7.3 on page 77.

Lastly, each cell of the state table that represents a valid state transition contains the name of the resulting state after the transition. The RTI Service States are listed in Section 7.7.4 on page 77.

If no valid state transition is listed in the cell at the intersection of a given service primitive and a given state, it is illegal to issue that service primitive while in that state.

7.7.2 Variables

Vrs Root RTI-SUI.

This variable, when TRUE, indicates that an RTI-SUI is the root of the transaction tree.

This variable is global to all contexts used by an RTI-SUI, its initial value is TRUE.

Vcs Client RTI-SUI.

This variable, when TRUE, indicates that an RTI-SUI is a client (superior) partner in a context tree.

This variable is local to a single context.

Vss Server RTI-SUI.

This variable, when TRUE, indicates that an RTI-SUI is a server (subordinate) partner in a context tree.

This variable is local to a single context.

Vtc Transactional Call.

This variable, when TRUE, indicates that the call is a transactional RPC (transaction mandatory or transaction optional within the scope of a global transaction).

This variable is local to a single context. It is set when an RTI-CALL-TASK.req is received.

Vte Transactions Enabled.

This variable, when TRUE, indicates that the RTI context was created with transactions enabled.

This variable is local to a single context. It is set when an RTI-ESTABLISH-CONTEXT.req is received.

7.7.3 Actions

[1] Actions related to issuing an RTI-ESTABLISH-CONTEXT request.

- Set *Vcs* to TRUE. When a context is created at the request of an RTI-SUI then that RTI-SUI behaves as a client RTI-SUI when using that context.
- If the value of Context-Type is TRANSACTION-ENABLED, set *Vte* to TRUE; otherwise, set it to FALSE.

[2] Actions related to issuing RTI-CALL-TASK request for a transactional RPC.

- Change the context to transactional and select the RTI Service Transactional State Table.

[3] Actions related to the receipt of an RTI-CALL-TASK indication.

- If the context being created is a non-transactional context then select the RTI Service Non-Transactional State Table.
- If the context being created is a transactional context then select the RTI Service Transactional State Table.
- Set *Vss* to TRUE. When an RTI-CALL-TASK indication is received by an RTI-SUI the context for that call has already been established. The RTI-SUI behaves as a server RTI-SUI when using that context.
- If the context is transactional, set *Vrs* to FALSE. A server RTI-SUI cannot be the root of a transaction tree if it is already part of a transaction tree.

7.7.4 States

I Idle.

CE Context Established.

CIP Call In Progress.

SP1 Start of Phase 1.

EP1 End of Phase 1.

SP2 Start of Phase 2.

EP2 End of Phase 2.

RBP Rollback in Progress.

RBC Rollback Complete.

7.7.5 State Tables

The RTI Service State Tables are listed in Table 7-8 and Table 7-9 on page 79.

Table 7-8 RTI Service Non-transactional State Table

Event	Pre.	I	CE	CIP
RTI-ESTABLISH-CONTEXT.req		[1] CE		
RTI-CALL-TASK.req	*Vcs* *Vte* *Vtc* *Vcs* !*Vtc*		[2] CIP CIP	
RTI-CALL-TASK.ind		[3] CIP		
RTI-CANCEL-CALL.req	*Vcs*			CIP
RTI-CANCEL-CALL.ind	*Vss*			CIP
RTI-CALL-FAILURE.ind	*Vcs*			CE / I †
RTI-CALL-RESULT.req	*Vss*			I
RTI-CALL-RESULT.ind	*Vcs*			CE
RTI-RELEASE-CONTEXT.req	*Vcs*		I	
RTI-RELEASE-CONTEXT.ind	*Vcs*		I	

Note:

† State transition depends on severity of error. See Section 7.4.4 on page 60.

This is the initial state table for all client and server RTI-PMs.

Service primitives shown in bold in the following table affect all transactional contexts.

Table 7-9 RTI Service Transactional State Table

Event	Pre.	CE	CIP	SP1	EP1	SP2	EP2	RBP	RBC
RTI-CALL-TASK.req	*Vcs*	CIP							
RTI-CALL-TASK.ind		CIP ‡							
RTI-CANCEL-CALL.req	*Vcs*		CIP						
RTI-CANCEL-CALL.ind	*Vss*		CIP						
RTI-CALL-FAILURE.ind	*Vcs*		CE / I †						
RTI-CALL-RESULT.req	*Vss*		CE						
RTI-CALL-RESULT.ind	*Vcs*		CE						
RTI-HEURISTIC-REPORT.ind	*Vcs*					SP2	EP2	RBP	RBC
RTI-ROLLBACK-TRANS.req		RBP	*Vss* RBP	RBP					
RTI-ROLLBACK-TRANS.ind		RBP	*Vss* RBP	RBP	RBP				
RTI-END-TRANS.req	*Vrs*	SP1							
RTI-PREPARE-TRANS.ind	*Vss*	SP1							
RTI-TRANS-READY.req	*Vss*			EP1					
RTI-COMMIT-TRANS.ind	*Vss*				SP2				
RTI-TRANS-READY.ind	*Vrs*			EP1					
RTI-COMMIT-TRANS.req	*Vrs*				SP2				
RTI-COMMIT-TRANS.ind	*Vrs*					SP2 ♦			
RTI-TRANS-DONE.req						EP2		RBC	
RTI-TRANS-COMPLETE.ind							CE / I §		CE / I §

Notes:

† State transition depends on severity of error. See Section 7.4.4 on page 60. If new state is I (Idle), select the RTI Service Non-Transactional State Table.

‡ Note that even when a non-transactional RTI-CALL-TASK.ind is received, the RTI context remains transactional for the purposes of reading Table 7-8 on page 78 and Table 7-9. However, the manager function is initiated in non-transaction mode.

♦ It may appear that the RTI-COMMIT-TRANS.ind for the Root RTI-SUI is not needed. However, this indication preserves a good mapping to OSI TP. Furthermore, depending on the TM, CRM and OSI TP implementations, its absence could imply a heuristic hazard condition for the ROOT RTI-SUI.

§ Select the RTI Service Non-Transactional State Table. If there are active context handles associated with the context, the new state is CE (Context Established). Otherwise, the new state is I (Idle).

Chapter 8

RTI Protocol Machine

This chapter describes the RTI Protocol Machine (RTI-PM). It does this by identifying the externally-defined supportive services used (OSI TP services), and describes:

- use of supportive services
- TP Services
- DC-ASE services
- RPC-ASE services
- RTI-MACF procedures
- RTI-APDU concatenation rules
- sequencing rules and state tables.

All sequencing rules are implicit in the state tables.

8.1 Use of Supportive Services

The RTI-PM makes use of the services provided by the OSI TPPM (TP services), the DC-ASE (DC services) and the RPC-ASE (RPC services). This section describes how RTI-PM makes use of those supportive services.

8.1.1 Relationship to Other Services

The RTI services provided by the RTI-PM are mapped onto the services provided by the OSI TPPM, DC-ASE and the RPC-ASE. These mappings are summarised in Table 8-1 below and Table 8-2 on page 83.

Service primitives shown in **bold** have complex mapping rules. Consult the appropriate RTI Service Primitive definition for details

Table 8-1 RTI Service Primitive Mapping Summary (Client)

RTI Service Primitives	TP Service Primitives	DC-ASE Service Primitives	RPC-ASE Service Primitives
RTI-ESTABLISH-CONTEXT.req		**DC-BEGIN-DIALOGUE.req**	
RTI-CALL-TASK.req	**TP-BEGIN-TRANSACTION.req**	**DC-BEGIN-DIALOGUE.req**	RPC-REQUEST.req
RTI-CANCEL-CALL.req			RPC-REMOTE-ALERT.req
RTI-CALL-FAILURE.ind	**TP-U-ABORT.ind** **TP-P-ABORT.ind** **TP-BEGIN-DIALOGUE.cnf** † **TP-ROLLBACK.ind**	**DC-REJECT-DIALOGUE.ind**	RPC-FAULT.ind
RTI-CALL-RESULT.ind			RPC-RESPONSE.ind
RTI-RELEASE-CONTEXT.req	TP-END-DIALOGUE.req		
RTI-RELEASE-CONTEXT.ind	**TP-U-ABORT.ind** **TP-P-ABORT.ind** **TP-BEGIN-DIALOGUE.cnf** †	**DC-REJECT-DIALOGUE.ind**	RPC-SHUTDOWN.ind
RTI-HEURISTIC-REPORT.ind	TP-HEURISTIC-REPORT.ind		
RTI-ROLLBACK-TRANS.req	TP-ROLLBACK.req		
RTI-ROLLBACK-TRANS.ind	**TP-ROLLBACK.ind** **TP-P-ABORT.ind**		
RTI-END-TRANS.req	**TP-DEFERRED-END-DIALOGUE.req** TP-PREPARE.req		
RTI-TRANS-READY.req	TP-COMMIT.req		
RTI-COMMIT-TRANS.ind	TP-COMMIT.ind		
RTI-TRANS-DONE.req	**TP-DONE.req** **TP-U-ABORT.req**		
RTI-TRANS-COMPLETE.ind	**TP-COMMIT-COMPLETE.ind** **TP-ROLLBACK-COMPLETE.ind**		

Notes:

† Only when Result parameter is set to "rejected(provider)".

Table 8-2 RTI Service Primitive Mapping Summary (Server)

RTI Service Primitives	TP Service Primitives	DC-ASE Service Primitives	RPC-ASE Service Primitives
RTI-CALL-TASK.ind			RPC-REQUEST.ind
RTI-CANCEL-CALL.ind			RPC-REMOTE-ALERT.ind
RTI-CALL-RESULT.req			RPC-RESPONSE.req
RTI-ROLLBACK-TRANS.req	TP-ROLLBACK.req		
RTI-ROLLBACK-TRANS.ind	**TP-ROLLBACK.ind** **TP-P-ABORT.ind**		
RTI-PREPARE-TRANS.ind	TP-PREPARE.ind		
RTI-TRANS-READY.ind	TP-READY.ind		
RTI-COMMIT-TRANS.req	TP-COMMIT.req		
RTI-COMMIT-TRANS.ind	TP-COMMIT.ind		
RTI-TRANS-DONE.req	TP-DONE.req		
RTI-TRANS-COMPLETE.ind	TP-COMMIT-COMPLETE.ind TP-ROLLBACK-COMPLETE.ind		

The following sections define the mapping of parameters on RTI service primitives onto the parameters of supportive services (TP, DC-ASE and RPC-ASE).

8.1.2 Mapping RTI-ESTABLISH-CONTEXT

There is no direct protocol flow initiated by the RTI-ESTABLISH-CONTEXT request.

The parameters of the RTI-ESTABLISH-CONTEXT request are mapped onto the parameters of the DC-BEGIN-DIALOGUE request by the client RTI-PM as follows:

RTI-AP-Title
: Maps to RTI-AP-Title on the DC-BEGIN-DIALOGUE request.

RTI-AE-Qualifier
: Maps to RTI-AE-Qualifier on the DC-BEGIN-DIALOGUE request.

Object-UUID
: Maps to Object-UUID on both the DC-BEGIN-DIALOGUE and the RPC-REQUEST requests.

Client-Name
: Maps to Client-Name on the DC-BEGIN-DIALOGUE request.

Client-Authenticator-Type
: Maps to Client-Authenticator-Type on the DC-BEGIN-DIALOGUE request.

Client-Authenticator
: Maps to Client-Authenticator on the DC-BEGIN-DIALOGUE request.

Interface-UUID
: Maps to Interface-UUID on the DC-BEGIN-DIALOGUE request.

Interface-Version-Major
: Maps to Interface-Version-Major on the DC-BEGIN-DIALOGUE request.

Interface-Version-Minor
: Maps to Interface-Version-Minor on the DC-BEGIN-DIALOGUE request.

Context-Type
: Maps to Context-Type on the DC-BEGIN-DIALOGUE request.

8.1.3 Mapping RTI-CALL-TASK

The parameters of the RTI-CALL-TASK request are mapped onto the RPC-REQUEST request and DC-BEGIN-DIALOGUE request by the client RTI-PM as follows:

Interface-UUID
: Maps to Interface-UUID on the RPC-REQUEST request. Also maps onto Interface-UUID of the DC-BEGIN-DIALOGUE request if this is the first RTI-CALL-TASK request within a particular context.

Interface-Version-Major
: Maps to Interface-Version-Major on the RPC-REQUEST request. Also maps onto Interface-Version-Major of the DC-BEGIN-DIALOGUE request if this is the first RTI-CALL-TASK request within a particular context.

Interface-Version-Minor
: Maps to Interface-Version-Minor on the RPC-REQUEST request. Also maps onto Interface-Version-Minor of the DC-BEGIN-DIALOGUE request if this is the first RTI-CALL-TASK request within a particular context.

Transaction-Attribute
: Maps to Transaction-Attribute on the RPC-REQUEST request.

Operation-Number
: Maps to Operation-Number on the RPC-REQUEST request.

Arguments
: Maps to Stub-Data on the RPC-REQUEST request.

The parameters of the DC-BEGIN-DIALOGUE indication and RPC-REQUEST indication (or indications) are mapped onto the parameters of the RTI-CALL-TASK indication by the server RTI-PM as follows:

Interface-UUID (from RPC-REQUEST.ind)
: Maps to Interface-UUID on the RTI-CALL-TASK indication.

Interface-Version-Major (from RPC-REQUEST.ind)
: Maps to Interface-Version-Major on the RTI-CALL-TASK indication.

Interface-Version-Minor (from RPC-REQUEST.ind)
: Maps to Interface-Version-Minor on the RTI-CALL-TASK indication.

Object-UUID (from RPC-REQUEST.ind)
: Maps to Object-UUID on the RTI-CALL-TASK indication.

Transaction-Attribute (from RPC-REQUEST.ind)
: Maps to Transaction-Attribute on the the RTI-CALL-TASK indication.

Operation-Number (from RPC-REQUEST.ind)
: Maps to Operation-Number on the RTI-CALL-TASK indication.

Stub-Data (from RPC-REQUEST.ind)
: Maps to Arguments on the RTI-CALL-TASK indication.

Client-Name (from DC-BEGIN-DIALOGUE.ind)
: Maps to Client-Name on the RTI-CALL-TASK indication.

Client-Authenticator-Type (from DC-BEGIN-DIALOGUE.ind)
: Maps to Client-Authenticator-Type on the RTI-CALL-TASK indication.

Client-Authenticator (from DC-BEGIN-DIALOGUE.ind)
Maps to Client-Authenticator on the RTI-CALL-TASK indication.

Context-Type (from DC-BEGIN-DIALOGUE.ind)
Maps to Context-Type on the RTI-CALL-TASK indication.

8.1.4 Mapping RTI-CANCEL-CALL

No parameter mapping.

8.1.5 Mapping RTI-CALL-FAILURE

The parameter Reason on the RPC-FAULT indication is mapped directly to the parameter Reason with the same name on the RTI-CALL-FAILURE indication by the client RTI-PM.

The parameter Diagnostic of the TP-P-ABORT indication is mapped onto the parameter Reason of the RTI-CALL-FAILURE indication by the client RTI-PM as shown in Table 8-3.

Table 8-3 Mapping of TP-P-ABORT Diagnostic

Diagnostic	Reason
permanent-failure	PERMANENT-COMMUNICATION-FAILURE
transient-failure	TRANSIENT-COMMUNICATION-FAILURE
begin-transaction-reject	not used
protocol-error	PROTOCOL-MACHINE-FAILURE

The reason code PROTOCOL-MACHINE-FAILURE is mapped onto the parameter Reason of the RTI-CALL-FAILURE indication by the client RTI-PM when a TP-U-ABORT indication results in an RTI-CALL-FAILURE indication.

When the parameter Result is "rejected(provider)" on the TP-BEGIN-DIALOGUE confirmation, the parameter Diagnostic is mapped onto the parameter Reason of the RTI-CALL-FAILURE indication by the RTI-PM as shown in Table 8-4.

Table 8-4 Mapping of TP-BEGIN-DIALOGUE.cnf Diagnostic

Diagnostic	Reason
recipient-unknown	RTI-SERVICE-UNKNOWN
recipient-tpsu-title-unknown	INTERFACE-UNKNOWN
tpsu-not-available(permanent)	INTERFACE-PERMANENTLY-UNAVAILABLE
tpsu-not-available(transient)	INTERFACE-TEMPORARILY-UNAVAILABLE
functional-unit-not-supported	CONTEXT-TYPE-NOT-SUPPORTED
no-reason-given	REASON-NOT-SPECIFIED

The parameter Reason-Code on the DC-REJECT-DIALOGUE indication maps directly to the parameter Reason with the same name on the RTI-CALL-FAILURE indication by the client RTI-PM.

The reason code ROLLBACK-IN-PROGRESS is mapped onto the parameter Reason of the RTI-CALL-FAILURE indication by the client RTI-PM when a TP-ROLLBACK indication causes an RTI-CALL-FAILURE indication.

8.1.6 Mapping RTI-CALL-RESULT

The parameter of the RTI-CALL-RESULT request is mapped onto the parameters of RPC-RESPONSE request data by the server RTI-PM as follows:

Arguments
: Maps to Stub-Data on the RPC-RESPONSE request.

The parameters on the RPC-RESPONSE indication are mapped onto the parameters of the RTI-CALL-RESULT indication by the client RTI-PM as follows:

Stub-Data
: Maps to Arguments on the RTI-CALL-RESULT indication.

8.1.7 Mapping RTI-RELEASE-CONTEXT

No parameter mapping.

8.1.8 Mapping RTI-HEURISTIC-REPORT

The parameters of the TP-HEURISTIC-REPORT indication are mapped onto the parameters of the RTI-HEURISTIC-REPORT indication by the client RTI-PM as follows:

Heuristic Report
: Maps to Diagnostic on the RTI-HEURISTIC-REPORT indication.

8.1.9 Mapping RTI-ROLLBACK-TRANS

No parameter mapping.

8.1.10 Mapping RTI-END-TRANS

No parameter mapping.

8.1.11 Mapping RTI-PREPARE-TRANS

No parameter mapping.

8.1.12 Mapping RTI-TRANS-READY

No parameter mapping.

8.1.13 Mapping RTI-COMMIT-TRANS

No parameter mapping.

8.1.14 Mapping RTI-TRANS-DONE

The parameters of RTI-TRANS-DONE request are mapped onto the parameters of the TP-DONE request as follows:

Heuristic-Report
: Maps to Heuristic-Report on the TP-DONE request.

8.1.15 Mapping RTI-TRANS-COMPLETE

No parameter mapping.

8.2 TP Services

TP services are defined in the OSI TP Service standard.

Table 8-5 identifies the OSI TP services used by the RTI-PM. Table 8-6 identifies the OSI TP Services not used by the RTI-PM. All services used are part of the OSI TP Dialogue and Commit functional units. No services from any other OSI TP functional unit are used.

All TP APDUs are carried as specified in Clauses 11.1 and 11.2 of the OSI TP Protocol standard. No additional rules are required.

Table 8-5 OSI TP Services Used by RTI-PM

Services	Request	Indication	Response	Confirmation
TP-BEGIN-DIALOGUE Confirmation set to negative	X	X		
TP-BEGIN-DIALOGUE Result set to rejected(provider)				X
TP-BEGIN-DIALOGUE Result set to rejected(user)			X	X
TP-END-DIALOGUE	X	X		
TP-HEURISTIC-REPORT		X		
TP-U-ABORT	X	X		
TP-P-ABORT		X		
TP-DEFERRED-END-DIALOGUE	X	X		
TP-BEGIN-TRANSACTION	X	X		X
TP-COMMIT	X	X		
TP-DONE	X			
TP-COMMIT-COMPLETE		X		
TP-PREPARE	X	X		
TP-READY		X		
TP-ROLLBACK	X	X		
TP-ROLLBACK-COMPLETE		X		

Table 8-6 OSI TP Services Not Used by RTI-PM

TP-U-ERROR
TP-GRANT-CONTROL
TP-REQUEST-CONTROL
TP-HANDSHAKE
TP-HANDSHAKE-AND-GRANT-CONTROL
TP-DEFERRED-GRANT-CONTROL

8.3 DC-ASE Services

This section defines the DC-ASE services and specifies the structure and encodings of the DC-ASE APDUs. The DC-ASE is an Application Service Element (ASE) that provides service primitives for assisting in dialogue establishment.

The general documentation conventions used here are the same as those specified in Section 7.1 on page 48.

8.3.1 Service Primitives

The DC-ASE services are invoked by using the service primitives listed in Table 8-7.

Table 8-7 DC-ASE Service Primitives

Service Primitives	Client	Server	See
DC-BEGIN-DIALOGUE	req	ind	Section 8.3.2.
DC-REJECT-DIALOGUE	ind	req	Section 8.3.3 on page 90.

Section 8.3.2 to Section 8.3.3 on page 90 inclusive define the service primitives of the DC-ASE.

8.3.2 DC-BEGIN-DIALOGUE request and indication

Function

The DC-BEGIN-DIALOGUE request and indication are used at dialogue establishment time to pass information between the client and the server RTI-PM. This information is used by the server RTI-PM to determine whether to accept or reject the incoming dialogue.

Parameters

Parameter Name	Req	Ind
Protocol-Version	M	M(=)
RTI-AP-Title	M	M(=)
RTI-AE-Qualifier	U	U(=)
Client-Name	M	M(=)
Client-Authenticator-Type	M	M(=)
Client-Authenticator	M	M(=)
Interface-UUID	M	M(=)
Interface-Version-Major	M	M(=)
Interface-Version-Minor	M	M(=)
Object-UUID	O	O(=)
Context-Type	M	M(=)

Protocol-Version
: This parameter specifies the RTI protocol version number.

RTI-AP-Title
: This parameter specifies the RTI application process title.

RTI-AE-Qualifier
: This parameter specifies the AE-Qualifier in which this server RTI-API resides. This parameter is optional.

Client-Name
This parameter carries the principal's name.

Client-Authenticator-Type
This parameter carries the security mechanism type.

Client-Authenticator
This parameter carries the authentication data.

Interface-UUID
This parameter specifies the UUID of the server interface for establishing this dialogue.

Interface-Version-Major
This parameter specifies the major version number of the server interface identified by the Interface-UUID.

Interface-Version-Minor
This parameter specifies the minor version number of the server interface identified by the Interface-UUID.

Object-UUID
This parameter specifies an (optional) object UUID for establishing this dialogue.

Context-Type
This parameter specifies the context type to be used.

Usage

The DC-BEGIN-DIALOGUE request may be issued only by the client RTI-PM.

The DC-BEGIN-DIALOGUE indication may be issued only by the server DC-ASE.

8.3.3 DC-REJECT-DIALOGUE request and indication

Function

The DC-REJECT-DIALOGUE request and indication are used at dialogue establishment time to allow the server RTI-PM to reject a dialogue.

Only a server RTI-PM can issue a DC-REJECT-DIALOGUE request.

This request cannot be issued on a dialogue if the server RTI-PM has previously issued other requests on that dialogue. The server RTI-PM cannot issue any other requests on a dialogue once the DC-REJECT-DIALOGUE request has been issued. That is, this must be the first and last request issued on a dialogue.

Parameters

Parameter Name	Req	Ind
Reason-Code	M	M(=)
Protocol-Versions	C	C(=)

Reason-Code
This parameter specifies a reason code. Reason-Code can take one of the following values:

PROTOCOL-VERSION-NOT-SUPPORTED
the requested RTI protocol version is not supported.

INTERFACE-PERMANENTLY-UNAVAILABLE
: the server RTI-API specified on the DC-BEGIN-DIALOGUE request supports the specified Interface but does not support the Interface Version specified.

INTERFACE-TEMPORARILY-UNAVAILABLE
: the server RTI-API specified on the DC-BEGIN-DIALOGUE request supports the requested Interface and Interface Version but does not have sufficient resources at this time to allow access.

REASON-NOT-SPECIFIED
: dialogue rejected for any other reason.

Protocol-Versions
: If the reason-code has the value Protocol-Versions-Not-Supported, then this parameter specifies all the Protocol-Versions supported by the server RTI-PM.

Usage

The DC-REJECT-DIALOGUE request may be issued only by the server RTI-PM.

The DC-REJECT-DIALOGUE indication may be issued only by the client DC-ASE.

8.3.4 Protocol Procedure

The protocol procedures for the DC-ASE are as follows:

DC-BEGIN-DIALOGUE request
: The DC-ASE issues a TP-BEGIN-DIALOGUE request.

TP-BEGIN-DIALOGUE indication
: The DC-ASE issues a DC-BEGIN-DIALOGUE indication.

DC-REJECT-DIALOGUE request
: The DCE-ASE issues a TP-BEGIN-DIALOGUE response with the Result parameter set to "rejected(user)".

TP-BEGIN-DIALOGUE confirmation
: TP-BEGIN-DIALOGUE confirmation is mapped to the DC-ASE only when the Result parameter is set to "rejected(user)". When the Result parameter is set to "rejected(user)" the DC-ASE issues a DC-REJECT-DIALOGUE indication.

8.3.5 Parameter Mappings

Table 8-8 on page 92 describes the mapping of the parameters of the DC-BEGIN-DIALOGUE request and indication service primitives. Parameters of the DC-BEGIN-DIALOGUE request map to parameters of the TP-BEGIN-DIALOGUE request and conditionally to fields of the DC-begin-dialogue APDU. Conversely, fields of the DC-begin-dialogue APDU and parameters of the TP-BEGIN-DIALOGUE indication map to parameters of the DC-BEGIN-DIALOGUE indication.

Table 8-8 DC-BEGIN-DIALOGUE Parameter Mapping

DC-BEGIN-DIALOGUE Parameter	TP-BEGIN-DIALOGUE Parameter	DC-BEGIN-DIALOGUE APDU field
Protocol-Version	User-Data	protocol-version
RTI-AP-Title	Recipient-AP-Title	
RTI-AE-Qualifier	Recipient-AE-Qualifier	
Client-Name	User-Data	client-name
Client-Authenticator-Type	User-Data	client-authenticator-type
Client-Authenticator	User-Data	client-authenticator
Interface-UUID	Recipient-TPSU-Title	
Interface-Version-Major	Recipient-TPSU-Title	
Interface-Version-Minor	Recipient-TPSU-Title	
Object-UUID	Recipient-TPSU-Title	
Context-Type	Functional-Units	
	Application-Context-Name†	
	Confirmation‡	

Notes:

† Always set to ObjectID for RTI Transactional or Non-transactional context.

‡ Always set to "negative".

Table 8-9 describes the mapping of the parameters of the DC-REJECT-DIALOGUE request to the parameters of the TP-BEGIN-DIALOGUE response. Conversely, fields of the DC-REJECT-DIALOGUE APDU and parameters of the TP-BEGIN-DIALOGUE confirmation are mapped to the parameters of the DC-REJECT-DIALOGUE indication

Table 8-9 DC-REJECT-DIALOGUE Parameter Mapping

DC-REJECT-DIALOGUE Parameter	TP-BEGIN-DIALOGUE Parameter	DC-REJECT-DIALOGUE APDU fields
Reason-Code	User-Data	reason-code
Protocol-Versions	User-Data	protocol-versions
	Result†	
	Rollback‡	

Notes:

† Always set to "rejected(user)".

‡ Always set to "false".

Recipient-TPSU-Title

Recipient-TPSU-Title is constructed by concatenating the Interface-UUID parameter, a single space, the version-number and the Object-UUID. Recipient-TPSU-Title is encoded as a PrintableString. The version-number is formed by concatenating the Interface-Version-Major and Interface-Version-Minor separated by a single period. The total length of the Recipient-TPSU-Title is 84 characters: 36 characters for Interface-UUID, one space character as a delimiter, five characters (digits only) for Interface-Version-Major, a period, five characters (digits only) for Interface-Version-Minor and 36 characters for Object-UUID[3]. Each of the 5-digit numbers is a decimal representation of an unsigned 16-bit integer with a maximum value of 65,535.

Functional-Units

Functional-Units are derived as shown in Table 8-10.

Table 8-10 Context-Type and Functional-Units

Context-Type	Functional-Unit
Transaction Enabled	Dialogue, Shared Control, Commit, Unchained Transaction
Not Transaction Enabled	Dialogue, Shared Control

8.3.6 Structure and Encoding of APDUs

The abstract syntax of each DC-ASE APDU is specified using ASN.1 (see the referenced ASN.1 standard).

```
-- Dialogue Control ASE APDUs

DC-apdus

{iso(1) national-member-body(2) bsi(826) disc(0) xopen(1050)
txrpc(6) abstract-syntax(2) dc-apdu(1) version1(1)}

DEFINITIONS ::=

BEGIN

IMPORTS

-- Transaction Processing
Transaction-Processing-APDUs FROM
{joint-iso-ccitt transaction-processing(10) modules(1)
apdu-abstract-syntax(1)}

-- Directory Service
DistinguishedName RelativeDistinguishedName FROM
InformationFramework
{joint-iso-ccitt ds(5) modules(1) informationFrame(1)}
```

3. Note that if the Object-UUID is not supplied, it is simply omitted from the Recipient-TPSU-Title.

```
-- The following defines the type of the EXTERNAL
-- field, User-information exported by Transaction-Processing-APDUs

DC-begin-dialogue ::= [APPLICATION 1] IMPLICIT SEQUENCE
  {
   protocol-version         BIT STRING
             {version1 (0)}

   client-name              VisibleString,

   client-authenticator-type INTEGER
        {
            default-security       (1),
            customer-written-security   (2)
        }

   client-authenticator     CHOICE
        {
            VisibleString,
            OCTET STRING
        }

  }

DC-reject-dialogue ::=[APPLICATION 2]  IMPLICIT SEQUENCE
  {
     reason-code ENUMERATED
         {
            protocol-version-not-supported    (0),
            interface-permanently-unavailable (1),
            interface-temporarily-unavailable (2),
            reason-not-specified              (3)
         }

     protocol-versions   BIT STRING OPTIONAL
                    {version1 (0)}

  }

END
```

8.4 RPC-ASE Services

The RPC-ASE implements the protocol described in the referenced X/Open **DCE RPC** specification.

The X/Open **DCE RPC** specification specifies in architectural terms the well defined communication capabilities of RPC. The communication capabilities provided by RPC are referred to as the RPC Application Service Element (RPC-ASE) to maintain consistency within the RTI model provided in this specification. This protocol is also described in Chapter 11, Connection-oriented RPC Protocol Machines, of the X/Open **DCE RPC** specification. It is respecified here to provide consistent style and terminology with the rest of the TxRPC specification.

Fundamental to understanding the RPC-ASE is an understanding of the structure of the RPC-ASE APDUs and the rules that govern the flow of these APDUs. The structure of the APDUs is fully described in Section 12.6, Connection-oriented RPC PDUs, of the X/Open **DCE RPC** specification.

This section defines the rules for flowing the APDUs defined in the X/Open **DCE RPC** specification. It does so by defining the RPC-ASE service primitives for use by the RTI protocol machine and their usage. The usage and sequencing rules defined are consistent with the rules defined in the X/Open **DCE RPC** specification. The connection oriented protocol is used to flow APDUs over an OSI TP dialogue, rather than a transport level connection as used in the X/Open **DCE RPC** specification.

This section describes the RPC-ASE by defining service primitives, protocol procedures, parameter mappings, and sequencing rules.

This section also identifies the RPC-ASE APDUs of the X/Open **DCE RPC** specification used by this document.

8.4.1 Service Conventions

The Service description conventions used here are as close as possible to OSI service description conventions.

The RPC-ASE has two service primitive classes: request (req) and indication (ind). Requests are issued by the RPC-ASE service user (that is the RTI-PM) and received by the RPC-ASE. Indications are issued by the RPC-ASE and received by the RPC-ASE service user (that is the RTI-PM).

Typically, although not necessarily, a request issued by one RPC-ASE service user results in a corresponding indication received by another RPC-ASE service user. In certain situations, a request issued to the RPC-ASE is purely a local event and does not directly generate protocol flow. Similarly, an indication issued by the RPC-ASE may result from a local event, not directly from protocol flow.

The relationship to service description conventions used in the X/Open **DCE RPC** specification is explained in Appendix D.

For consistency with the conventions used in other sections of this document and naming conventions for international standards, the service primitive names do not match the names for events and actions in the X/Open **DCE RPC** specification. The relationship to the event and action names in the X/Open **DCE RPC** specification are described in Appendix D.

8.4.2 Service Primitives

The RPC-ASE service is invoked using a sequence of RPC-ASE service primitives. Table 8-11 summarises the RPC-ASE service primitives. This table contains the service primitive name, primitive type at client (req or ind), primitive type at server (req or ind), and the section of this document that fully describes the service primitive.

Table 8-11 RPC-ASE Service Primitives Summary

Primitive Name	Client	Server	See
RPC-REQUEST	req	ind	Section 8.4.3.
RPC-RESPONSE	ind	req	Section 8.4.4 on page 98.
RPC-ORPHANED	req	ind	Section 8.4.5 on page 98.
RPC-REMOTE-ALERT	req	ind	Section 8.4.6 on page 99.
RPC-FAULT	ind	req	Section 8.4.7 on page 99.
RPC-NO-CONN	req	req	Section 8.4.8 on page 100.
RPC-DONE	ind	ind	Section 8.4.9 on page 101.
RPC-SHUTDOWN	ind	req	Section 8.4.10 on page 101.

Section 8.4.3 to Section 8.4.10 on page 101 define the service primitives.

8.4.3 RPC-REQUEST

Function

The RPC-REQUEST request is issued by the client RTI-PM to transmit an rpc_request APDU.

The RPC-REQUEST indication is issued by the server RPC-ASE to the server RTI-PM to indicate the contents of an rpc_request APDU.

Parameters

Parameter Name	Req	Ind
Interface-UUID	M	M(=)
Interface-Version-Major	M	M(=)
Interface-Version-Minor	M	M(=)
Object-UUID	O	O(=)
Transaction-Attribute	M	M(=)
Operation-Number	M	M (=)
Stub-Data	M	M (=)
Last-Frag	O	O (=)
Auth-Type	O	O (=)
Auth-Level	O	O (=)
Auth-Value	O	O (=)

Interface-UUID
: This parameter is supplied by the client RTI-PM. It indicates the server interface for the call.

Interface-Version-Major
: This parameter is supplied by the client RTI-PM. It indicates the major version number of the interface.

Interface-Version-Minor
: This parameter is supplied by the client RTI-PM. It indicates the minor version number of the interface.

Object-UUID
: This parameter is supplied by the client RTI-PM. It indicates the (optional) object UUID associated with the call.

Transaction-Attribute
: This parameter is supplied by the client RTI-PM. It indicates the transaction attribute (TRANSACTION-MANDATORY, TRANSACTION-OPTIONAL or TRANSACTION-NONE) of the call.

Operation-Number[4]
: This parameter is supplied by the client RTI-PM. It indicates the operation number for the call.

Stub-Data
: This parameter is supplied by the client RTI-PM. It contains the user data.

Last-Frag
: This parameter is supplied by the client RTI-PM. Its presence indicates the last rpc_request fragment for a call.

Auth-Type[5]
: This parameter is supplied by the client RTI-PM.

Auth-Level[6]
: This parameter is supplied by the client RTI-PM.

Auth-Value[7]
: This parameter is supplied by the client RTI-PM.

Usage

The RPC-REQUEST request may be issued only by the client RTI-PM.

The RPC-REQUEST indication may be issued only by the server RPC-ASE.

4. The Operation-Number (opn) is a 16-bit non-negative integer that identifies a particular operation within an interface. On a call to a remote operation, this field contains the number of the target operation within the interface. (Operations are numbered in the order in which they are defined in the interface, starting with 0.)
5. This parameter is not used in this version. It is reserved for future use.
6. This parameter is included for compatibility with future versions of RPC. It is not used in this version.
7. This parameter is not used in this version. It is reserved for future use.

8.4.4 RPC-RESPONSE

Function

The RPC-RESPONSE request is issued by the server RTI-PM to transmit an rpc_response APDU.

The RPC-RESPONSE indication is issued by the RPC-ASE to indicate an rpc_response APDU to the client RTI-PM.

Parameters

Parameter Name	Req	Ind
Stub-Data	M	M (=)
Last-Frag	O	O (=)

Stub-Data
: This parameter is supplied by the server RTI-PM. It contains the transaction context and the user data.

Last-Frag
: This parameter is supplied by the server RTI-PM. Its presence indicates the last rpc_response fragment for the call.

Usage

The RPC-RESPONSE request may be issued only by the server RTI-PM.

The RPC-RESPONSE indication may be issued only by the client RPC-ASE.

8.4.5 RPC-ORPHANED

This section defines the RPC-ORPHANED service primitive.

Function

The RPC-ORPHANED request is issued by the client RTI-PM to transmit an rpc_orphaned APDU.

The RPC-ORPHANED indication is issued by the server RPC-ASE to indicate an rpc_orphaned APDU to the server RTI-PM.

RPC-ORPHANED is used to abruptly terminate a call in progress.

Parameters

None.

Usage

The RPC-ORPHANED request may be issued only by the client RTI-PM. The RPC-ORPHANED request may be issued at most once for a call.

The RPC-ORPHANED indication may be issued only by the server RPC-ASE.

8.4.6 RPC-REMOTE-ALERT

This section defines the RPC-REMOTE-ALERT service primitive.

Function

The RPC-REMOTE-ALERT service primitive is issued by the client RTI-PM to transmit an rpc_remote_alert APDU.

The RPC-REMOTE-ALERT indication is issued by the server RPC-ASE to indicate an rpc_remote_alert apdu to the server RTI-PM.

Parameters

None

Usage

The RPC-REMOTE-ALERT request may be issued only by the client RTI-PM. The RPC-REMOTE-ALERT request may be issued multiple times.

The RPC-REMOTE-ALERT indication may be issued only by the server RPC-ASE.

8.4.7 RPC-FAULT

This section defines the RPC-FAULT service primitive.

Function

The RPC-FAULT request is issued by the server RTI-PM to transmit an rpc_fault APDU.

The RPC-FAULT indication is issued by the client RPC-ASE to indicate an rpc_fault APDU to the client RTI-PM.

Parameters

Parameter Name	Req	Ind
Reason	M	M (=)
DNE	O	O (=)
Stub-Data	O	O (=)

Reason

This parameter is supplied by the server RTI-PM. This parameter can be one of the following reasons. See Table 9-1 on page 137 for corresponding hexadecimal values.

- RPC-ACCESS-VIOLATION
- RPC-CANCEL
- RPC-FLOATING-DIVIDE-BY-ZERO
- RPC-FLOATING-ERROR
- RPC-FLOATING-OVERFLOW
- RPC-FLOATING-UNDERFLOW
- RPC-INSUFFICIENT-RESOURCES

- RPC-INTEGER-DIVIDE-BY-ZERO
- RPC-INTEGER-OVERFLOW
- RPC-INVALID-OPERATION-NUMBER
- RPC-INVOCATION-FAILURE
- RPC-MARSHALLING-ERROR
- RPC-PROTOCOL-ERROR
- RPC-REASON-NOT-SPECIFIED.

DNE
: This parameter is supplied by the server RTI-PM. Its presence indicates the call did not execute. Its absence indicates the call need not have executed.

Stub-Data[8]
: This parameter is supplied by the server RTI-PM. It contains optional stub data.

Usage

The RPC-FAULT request may be issued only by the server RTI-PM.

The RPC-FAULT indication may be issued only by the client RPC-ASE.

8.4.8 RPC-NO-CONN

Function

The RPC-NO-CONN request is issued by the RTI-PM to request the RPC-ASE to terminate any activities associated with the currently established context. The RPC-NO-CONN request generates no protocol flow.

Parameters

None.

Usage.

The RPC-NO-CONN request may be issued by the client RTI-PM or server RTI-PM. The RPC-NO-CONN request may be issued at any time.

8. This parameter is not used in this version. It is reserved for future use.

8.4.9 RPC-DONE

This section defines the RPC-DONE indication service primitive.

Function

The RPC-DONE indication is issued by the RPC-ASE to indicate to the RTI-PM its readiness for the next data fragment.[9]

Parameters

None.

Usage

The RPC-DONE indication may be issued either by the client RPC-ASE or server RPC-ASE.

8.4.10 RPC-SHUTDOWN

Function

The RPC-SHUTDOWN request is issued by the server RTI-PM to transmit an rpc_shutdown APDU.

The RPC-SHUTDOWN indication is issued by the client RPC-ASE to indicate an rpc_shutdown APDU to the client RTI-PM.

Parameters

None.

Usage

The RPC-SHUTDOWN request service primitive may be issued only by the server RTI-PM.

The RPC-SHUTDOWN indication service primitive may be issued only by the client RPC-ASE.

8.4.11 Protocol Procedures

A remote procedure call consists of a sequence of RPC-ASE APDUs. Each remote procedure call is assigned a call_id value by the client RPC-ASE when the first rpc_request APDU is transmitted. Uniqueness of call_id is defined in Section 6.1.3, Remote Procedure Calls, of the X/Open **DCE RPC** specification.

Unless otherwise specified, every APDU transmitted by the client RPC-ASE and server RPC-ASE for a given remote procedure call is assigned the same call_id value.

At any time an RPC-ASE has only one valid call_id value. Any APDUs with a call_id other than the valid call_id are discarded.

---

9. Issuance of the RPC-DONE indication does not indicate any information about the state of data being transmitted, it only indicates that the local RPC-ASE has fully received the data fragment and is ready to receive another. An implementation may choose a policy for issuing the RPC-DONE indication for proper resource utilisation. This service primitive does not appear in the X/Open **DCE RPC** specification. It was added to help clarify the operation of segmentation.

The call_id value is mapped to the **call_id** field of the RPC-ASE APDUs described in the X/Open **DCE RPC** specification.

Client Protocol Procedures

The client RPC-ASE protocol procedures are as follows:

RPC-REQUEST request

On the first RPC-REQUEST request for a remote procedure call the client RPC-ASE performs the following actions:

- determines a call_id and records it as the current call_id.
- sets the PFC_FIRST_FRAG flag.

If the Last-Frag parameter is present the RPC-ASE sets the PFC_LAST_FRAG flag.

The RPC-ASE transmits an rpc_request APDU.

If the Last-Frag parameter is not present the RPC-ASE issues an RPC-DONE indication.

RPC-REMOTE-ALERT request

The RPC-ASE transmits an rpc_remote_alert APDU.

The RPC-ASE queues an alert timeout timer.[10]

Once queued, an alert timeout timer is active until all operations related to the call are complete and the client state machine returns to the idle state. In other words, once an alert timeout timer has been queued in response to the RPC-REMOTE-ALERT request the call must must complete (including all response fragments) before the alert timer expires to avoid the actions of an alert timeout (issue RPC-FAULT.ind and transmit rpc_orphaned APDU).

RPC-ORPHANED request

The RPC-ASE transmits an rpc_orphaned APDU.

The current call_id is invalidated.

RPC-NO-CONN request

The RPC-ASE discards any APDUs queued for transmission.

The current call_id is invalidated.

rpc_response APDU

The RPC-ASE issues an RPC-RESPONSE indication with parameters as follows:

- If the PFC_LAST_FRAG flag is set the Last-Frag parameter is supplied.

rpc_fault APDU

The RPC-ASE issues an RPC-FAULT indication. If the flag PFC_DID_NOT_EXECUTE is set the DNE parameter is included on the indication.

The current call_id is invalidated.

rpc_shutdown APDU

The **call_id** field does not apply to the rpc_shutdown APDU.

10. The timeout period for alerts is implementation-specific.

The RPC-ASE issues an RPC-SHUTDOWN indication.

AlertTimeout
An AlertTimeout indicates that the server has not responded to a remote alert in the prescribed period of time. The RPC-ASE issues an RPC-FAULT indication to the RTI-PM with reason code RPC-PROTOCOL-ERROR. Then the RPC-ASE transmits an rpc_orphaned APDU.

The current call_id is invalidated.

Server Protocol Procedures

The server protocol procedures are as follows:

RPC-RESPONSE request
The RPC-ASE transmits an rpc_response APDU with flags as follows.

- On the first RPC_RESPONSE request for a call the RPC-ASE sets the PFC_FIRST_FRAG flag.
- If the Last-Frag parameter is present the RPC-ASE sets the PFC_LAST_FRAG flag.

If the Last-Frag parameter is not present the RPC-ASE issues an RPC-DONE indication.

If the Last-Frag parameter is present the current call_id is invalidated.

RPC-FAULT request
The RPC-ASE transmits an rpc_fault APDU. If the DNE parameter is present on the request the flag PFC_DID_NOT_EXECUTE is set by the RPC-ASE.

The current call_id is invalidated.

RPC-SHUTDOWN request
The RPC-ASE transmits an rpc_shutdown APDU.

The **call_id** field is not set for the rpc_shutdown APDU.

RPC-NO-CONN request
The RPC-ASE discards any APDUs queued for transmission.

The current call_id is invalidated.

rpc_request APDU
The RPC-ASE issues an RPC-REQUEST indication with parameters as follows:

- If the PFC_LAST_FRAG flag is set the Last-Frag parameter is supplied.

If the PFC_FIRST_FRAG flag is set the value of the **call_id** field is recorded as the current call_id.

rpc_orphaned APDU
The RPC-ASE issues an RPC-ORPHANED indication.

The current call_id is invalidated.

rpc_remote_alert APDU
The RPC-ASE issues an RPC-REMOTE-ALERT indication.

8.4.12 Mapping to Lower Layers

All RPC-ASE APDUs are mapped onto the P-Data service.

8.4.13 Parameter Mappings

The mapping between service primitive parameters and corresponding APDU fields is shown in the following tables.

Parameters on the RPC-REQUEST service primitive are mapped onto the rpc_request APDU as shown in Table 8-12.

Table 8-12 rpc_request APDU Mapping

Parameter	rpc_request APDU field
Interface-UUID	context_id†
Interface-Version-Major	context_id†
Interface-Version-Minor	context_id†
Object-UUID	object
Transaction-Attribute	transaction_indicator‡
Operation-Number	opnum
Stub-Data	stub data goes here
Last-Frag	PFC_LAST_FRAG
Auth-Type	auth_type
Auth-Level	auth_level
Auth-Value	auth_value

Notes:

† Interface-UUID, Interface-Version-Major and Interface-Version-Minor are passed indirectly via the context_id field of the rpc_request APDU. This field identifies a presentation context negotiated for the association.

‡ This field is in the RPC request header extension (see Section 8.4.14 on page 105). If there is a current transaction and the Transaction_Attribute is **transaction_mandatory** or **transaction_optional**, the value is transactional_rpc. Otherwise the value is non_transactional_rpc.

Parameters on the RPC-RESPONSE service primitive are mapped onto the rpc_response APDU as shown in Table 8-13.

Table 8-13 rpc_response APDU Mapping

Parameter	rpc_response APDU
Stub-Data	stub data goes here
Last-Frag	PFC_LAST_FRAG

Parameters on the RPC-FAULT service primitive are mapped onto the rpc_fault APDU as shown in Table 8-14 on page 105.

Table 8-14 rpc_fault APDU Mapping

Parameter	rpc_fault APDU
Reason	status
DNE	PFC_DID_NOT_EXECUTE
Stub-Data	stub data goes here

There is no parameter mapping for:

- rpc_shutdown APDU
- rpc_remote_alert APDU
- rpc_orphaned APDU.

8.4.14 Structure and Encoding of APDUs

The formal ASN.1 definition for the RPC-ASE APDUs is given below. The format and encoding of the RPC-ASE APDUs is defined in Section 12.6, Connection-oriented RPC PDUs, of the X/Open **DCE RPC** specification.

```
-- Remote Task Invocation RPC-ASE APDUs Version 1

RPC-apdus-1

{iso(1) national-member-body(2) bsi(826) disc(0) xopen(1050)
txrpc(6) abstract-syntax(2) rpc-apdu(2) version1(1)}

DEFINITIONS IMPLICIT TAGS ::=
BEGIN
RPC-ASE-apdu ::= OCTET STRING

-- The contents of the OCTET STRING can have one of six formats.
-- The six RPC-ASE-apdu formats are designated:
--  rpc_request APDU
--  rpc_response APDU
--  rpc_fault APDU
--  rpc_shutdown APDU
--  rpc_remote_alert APDU
--  rpc_orphaned APDU
-- The RPC-ASE-apdu formats are described in
-- Chapter 12.6 of the X/Open DCE RPC specification.

END
```

rpc_request APDU

The structure and encoding of the rpc_request APDU is defined in Section 12.6.4.9, The request PDU, of the X/Open **DCE RPC** specification. A header extension, defined below, is added after the standard rpc_request header (after the optional object field). This is followed by the stub data (8-octet aligned) and the optional authentication trailer.

The rpc_request APDU is distinguished by the value rpc_request in the PTYPE field.

The format of the header extension is described below in IDL notation.

```
typedef struct
{
    unsigned small rollback_indicator;
    unsigned small transaction_indicator;
} xoext;

typedef union switch (long) ext_hdr
{
    case
        1: xoext a_xoext;
} extended_header;

/* Rollback indicator value to indicate no rollback */

  #define no_rollback 0

/* Rollback indicator value to indicate rollback by server */

  #define rollback_by_server 1

/* transaction_indicator value to indicate not transactional */

  #define non_transactional_rpc 0

/* transaction_indicator value to indicate transactional */

  #define transactional_rpc 1
```

rpc_response APDU

The structure and encoding of the rpc_response APDU is defined in Section 12.6.4.10, The response PDU, of the X/Open **DCE RPC** specification. A header extension, defined in the description of the rpc_request APDU, is added after the standard rpc_response header (after the reserved field). This is followed by the stub data (8-octet aligned) and the optional authentication trailer.

The rpc_response APDU is distinguished by the value rpc_response in the PTYPE field.

rpc_fault APDU

The structure and encoding of the rpc_fault APDU is defined in Section 12.6.4.7, The fault PDU, of the X/Open **DCE RPC** specification.

The rpc_fault APDU is distinguished by the value rpc_fault in the PTYPE field.

rpc_shutdown APDU

The structure and encoding of the rpc_shutdown APDU is defined in Section 12.6.4.11, The shutdown PDU, of the X/Open **DCE RPC** specification.

The rpc_shutdown APDU is distinguished by the value rpc_shutdown in the PTYPE field.

rpc_remote_alert APDU

The structure and encoding of the rpc_remote_alert APDU is defined in Section 12.6.4.6, The cancel PDU, of the X/Open **DCE RPC** specification.

The rpc_remote_alert APDU is distinguished by the value rpc_remote_alert in the PTYPE field.

rpc_orphaned APDU

The structure and encoding of the rpc_orphaned APDU is defined in Section 12.6.4.8, The orphaned PDU, of the X/Open **DCE RPC** specification.

The rpc_orphaned APDU is distinguished by the value rpc_orphaned in the PTYPE field.

8.4.15 Sequencing

To simplify and enhance readability two state tables are defined, the RPC-ASE client state table and the RPC-ASE server state table. State transitions for the client RPC-ASE are described by the RPC-ASE client state table. State transitions for the server RPC-ASE are defined by the RPC-ASE server state table.

Table 8-19 on page 109 and Table 8-23 on page 111 (the state tables) define the allowed sequence of service events and protocol events for a given RPC-ASE.

A separate state is maintained for each context established by the RTI-PM. RPC-ASE service primitives relate to one context only.

The state tables specify preconditions that must be satisfied in order for individual events to be allowed in a given state. These preconditions are based on service primitive parameters and protocol fields.

The state tables also specify actions to be performed. These actions involve issuing service primitive indications and transmitting APDUs.

Table 8-15 identifies the states for Table 8-19 on page 109 and Table 8-23 on page 111.

Table 8-15 RPC-ASE States

State	Description
I	The idle state
RIP	Request in Progress
CIP	Call In Progress
RSP	Response in Progress

Client RPC-ASE

Sequencing rules for the client RPC-ASE are shown in the following tables.

Table 8-16 Client RPC-ASE Events

Event	Description
RPC-REQUEST.req	RPC-REQUEST request received
RPC-ORPHANED.req	RPC-ORPHANED request received
RPC-REMOTE-ALERT.req	RPC-REMOTE-ALERT request received
RPC-NO-CONN.req	RPC-NO-CONN request received
rpc_fault APDU	rpc_fault APDU received
rpc_response APDU	rpc_response APDU received
rpc_shutdown APDU	rpc_shutdown APDU received
AlertTimeout	AlertTimer timeout event occurred

Table 8-17 Client RPC-ASE Preconditions

Precondition	Description
plf	PFC_LAST_FRAG flag set
lfg	Last-Frag parameter present
pdn	PFC_DID_NOT_EXECUTE flag set

Table 8-18 Client RPC-ASE Actions

Action	Description
irft	Issue RPC-FAULT indication
irsp	Issue RPC-RESPONSE indication
irsd	Issue RPC-DONE indication
issr	Issue RPC-SHUTDOWN
tral	Transmit rpc_remote_alert APDU
treq	Transmit rpc_request APDU
torp	Transmit rpc_orphaned APDU
A1	Queue an alert timeout timer

Table 8-19 Client RPC-ASE State Table

Event	Precon	I	RIP	CIP	RSP
RPC-REQUEST.req	¬lfg	treq irsd RIP	treq irsd RIP		
RPC-REQUEST.req	lfg	treq CIP	treq CIP		
RPC-REMOTE-ALERT.req			A1 tral RIP	A1 tral CIP	A1 tral RSP
RPC-ORPHANED.req			torp I	torp I	torp I
rpc_response APDU	plf	I		irsp I	irsp I
rpc_response APDU	¬plf	I		irsp RSP	irsp RSP
rpc_fault APDU	¬pdn	I	irft I	irft I	irft I
rpc_fault APDU	pdn	I	irft I	irft I	
AlertTimeout		I	irft torp I	irft torp I	irft torp I
RPC-NO-CONN.req			I	I	I
rpc_shutdown APDU		issr I	issr RIP	issr CIP	

Server RPC-ASE

Sequencing rules for the server RPC-ASE are shown in the following tables.

Table 8-20 Server RPC-ASE Events

Event	Description
rpc_request APDU	rpc_request APDU received
rpc_remote_alert APDU	rpc_remote_alert APDU received
rpc_orphaned APDU	rpc_orphaned APDU received
RPC-NO-CONN.req	RPC-NO-CONN request received
RPC-FAULT.req	RPC-FAULT request received
RPC-RESPONSE.req	RPC-RESPONSE request received
RPC-SHUTDOWN.req	RPC-SHUTDOWN request received

Table 8-21 Server RPC-ASE Preconditions

Precondition	Description
lrf	PFC_LAST_FRAG flag set in received APDU
frf	PFC_FIRST_FRAG flag set in received APDU
lsf	The Last-Frag parameter is present
dne	The DNE parameter is present

Table 8-22 Server RPC-ASE Actions

Action	Description
ireq	Issue RPC-REQUEST indication
iral	Issue RPC-REMOTE-ALERT indication
iorp	Issue RPC-ORPHANED indication
irfd	Issue RPC-DONE indication
tflt	Transmit rpc_fault APDU
trsp	Transmit rpc_response APDU
tsht	Transmit rpc_shutdown APDU

Table 8-23 Server RPC-ASE State Table

Events	Precon	I	RIP	CIP	RSP
rpc_request APDU	frf & lrf	ireq CIP			
rpc_request APDU	frf & ¬lrf	ireq RIP			
rpc_request APDU	¬frf & lrf	I	ireq CIP		
rpc_request APDU	¬frf & ¬lrf	I	ireq RIP		
rpc_remote_alert APDU		I	iral RIP	iral CIP	iral RSP
rpc_orphaned APDU		I	iorp I	iorp I	iorp I
RPC-RESPONSE.req	lsf			trsp I	trsp I
RPC-RESPONSE.req	¬lsf			trsp irfd RSP	trsp irfd RSP
RPC-FAULT.req	¬dne			tflt I	tflt I
RPC-FAULT.req	dne		tflt I	tflt I	
RPC-NO-CONN.req			I	I	I
RPC-SHUTDOWN.req		tsht I			

8.5 RTI-MACF Procedures

The MACF procedures of the RTI Protocol Machine (RTI-PM) are as follows:

- RTI requests
- DC-ASE indications
- RPC-ASE indications
- TP indications
- internal events.

8.5.1 Rules

The rules used for the operation of the procedures are as follows:

- The actions for each procedure must be executed in the described order.
- The RTI-PM completely executes all actions pertinent to an event before accepting a new event. Furthermore, when an RTI-MACF procedure invokes the services of the lower layers (for example RPC-ASE), it is assumed that the entire processing is atomic. That is, the MACF procedure and all pertinent lower layer procedures are completely executed before a new event is accepted.
- Unless otherwise stated, all procedures describe the actions for both the transactional and non-transactional contexts.

8.5.2 Definitions

The following definitions are used in the descriptions of the RTI-PM procedures:

Request in progress
: For the client RTI-PM one or more RPC-REQUEST requests (but none with Last-Frag parameter) have been issued to the RPC-ASE.

  For the server RTI-PM, one or more RPC-REQUEST indications have been received from the RPC-ASE but none with Last-Frag parameter has been received.

Call in progress
: For the client RTI-PM, an RPC-REQUEST request with Last-Frag parameter has been issued to the RPC-ASE. The client RTI-PM may have received one or more RPC-RESPONSE indications but none has been received with the Last-Frag parameter present.

  For the server RTI-PM, an RPC-REQUEST request with Last-Frag parameter has been received from the RPC-ASE but no RPC-RESPONSE request has been issued to the RPC-ASE.

Response in progress
: For the server RTI-PM, one or more RPC-RESPONSE requests have been issued to the RPC-ASE but none with Last-Frag parameter has been issued.

Segmentation required
: If the data supplied as the Arguments parameter of the RTI-CALL-TASK or RTI-CALL-RESULT request is larger than the maximum RPC-ASE segment size, the segmentation of data is required. The RTI-PM is responsible for performing the segmentation.

  On the first attempt to issue any of the data transfer primitives of the RPC-ASE to transfer the call or result data, the term *segmentation required* implies that the size of the user data is larger than the maximum segment size. On any subsequent attempts to transfer the

remaining portions of the data, the term *segmentation required* implies that the remaining data is larger than the maximum segment.

Segmentation storage

This term is applicable only when segmentation is required.

For the client RTI-PM, the segmentation storage is the internal workspace used by the RTI-PM to store the user data supplied by the Arguments parameter on the RTI-CALL-TASK request; or to assemble the data to be provided to the client RTI-SUI on the Arguments parameter of the RTI-CALL-RESULT indication.

For the server RTI-PM, the segmentation storage is the internal workspace used by the RTI-PM to assemble the user data to be provided to the server RTI-SUI on the Arguments parameter of the RTI-CALL-TASK indication; or to store the user data supplied by the Arguments parameter on the RTI-CALL-RESULT request.

The RTI-PM maintains a separate segmentation storage for each context.

8.5.3 RTI Request Procedures

The RTI-PM MACF procedures for the RTI requests are as follows:

RTI-ESTABLISH-CONTEXT request

The RTI-PM remembers the parameters of the RTI-ESTABLISH-CONTEXT request for use when the DC-BEGIN-DIALOGUE request is issued.

RTI-CALL-TASK request

If this is the first RTI-CALL-TASK request for a particular context, the RTI-PM issues a DC-BEGIN-DIALOGUE request with the parameters from the previous RTI-ESTABLISH-CONTEXT request and the RTI-CALL-TASK request.

If the call is transactional and the context is not already transactional, the context is made transactional and the dialogue is included in the transaction; the RTI-PM issues a TP-BEGIN-TRANSACTION request.

If segmentation is required, the RTI-PM:

1. saves the user data supplied by the Arguments parameter into the segmentation storage
2. issues an RPC-REQUEST request without Last-Frag parameter to transmit the first segment of the call data.

If segmentation is not required, the RTI-PM issues an RPC-REQUEST request with Last-Frag parameter to transmit the entire call data.

RTI-CANCEL-CALL request

The RTI-PM issues an RPC-REMOTE-ALERT request.

RTI-CALL-RESULT request

For non-transactional context, if the dialogue is already aborted or orphaned, the RTI-PM ignores this request. In all other cases, the RTI-PM performs the following tasks:

If segmentation is required:

1. It saves the user data supplied by the Arguments parameter into the segmentation storage.
2. It issues an RPC-RESPONSE request without the Last-Frag parameter to transmit the first segment of the call result data.

If segmentation is not required:

1. It issues an RPC-RESPONSE request with the Last-Frag parameter present to transmit the entire call result data.

RTI-RELEASE-CONTEXT request

The RTI-PM issues a TP-END-DIALOGUE request with the Confirmation parameter set to "false" and releases the non-transactional context.

RTI-ROLLBACK-TRANS request

This procedure applies to the transactional context only. The server RTI-PM performs the following tasks:

1. If the first call is completed and a subsequent request is in progress, it issues an RPC-FAULT request with the DNE parameter present and Reason parameter set to RPC-REASON-NOT-SPECIFIED.[11]
2. If a response is in progress, it issues an RPC-FAULT request without the DNE parameter and Reason parameter set to RPC-REASON-NOT-SPECIFIED.

The RTI-PM issues a TP-ROLLBACK request.

RTI-END-TRANS request

This procedure applies to the transactional context only. The server RTI-PM performs the following tasks:

1. If there are no active context handles associated with the context, issue a TP-DEFERRED-END-DIALOGUE request.
2. Issue a TP-PREPARE request.

11. The server RTI-PM can never receive this request while it is in the request in progress state with respect to the first call.

RTI-TRANS-READY request

This procedure applies to the transactional context only.

The RTI-PM issues a TP-COMMIT request.

RTI-TRANS-DONE request

This procedure applies to the transactional context only.

The RTI-PM performs the following actions:

1. If processing a transaction rollback on the client side, and there are no active context handles associated with the transactional context, it issues a TP-U-ABORT request. This is so that the dialogue is terminated when the rollback processing is completed.
2. It issues a TP-DONE request.

8.5.4 DC-ASE Indication Procedures

The procedures for the DC-ASE indications are as follows:

DC-BEGIN-DIALOGUE indication

This indication is received by the server RTI-PM.

If the RTI-PM accepts the dialogue, no action is required.

If the RTI-PM rejects the dialogue, it issues the DC-REJECT-DIALOGUE request with one of the following reason codes:

protocol-version-not-supported
: The requested RTI protocol version is not supported. In this case, the protocol-versions parameter of the DC-REJECT-DIALOGUE request indicates the protocol versions supported by the server RTI-PM.

interface-permanently-unavailable
: The version of the interface does not match the value of the interface-version parameter.

interface-temporarily-unavailable
: The interface is known and its version matches the value of the interface-version parameter but the interface is temporarily unavailable.

reason-not-specified
: Dialogue rejected for any other reason.

DC-REJECT-DIALOGUE indication

This indication is received by the client RTI-PM when a request or a call is in progress.[12]

The RTI-PM performs the following actions:

1. It issues an RTI-CALL-FAILURE indication. The Reason-Code parameter of the DC-REJECT-DIALOGUE indication is mapped onto the Reason parameter of the RTI-CALL-FAILURE indication.

12. Since the dialogue is established as part of the first call, this indication can only be received while a request or a call is in progress.

2. For non-transactional context, it releases the context and issues an RTI-RELEASE-CONTEXT indication.
3. It issues an RPC-NO-CONN request.

8.5.5 RPC-ASE Indication Procedures

The procedures for RPC-ASE indication are as follows:

RPC-REQUEST indication

This indication indicates that one segment of the call request data has been received.

If the Last-Frag parameter is absent, the RTI-PM saves the call request data received on this indication in the segmentation storage.

If the Last-Frag parameter is present:

1. The server RTI-PM passes the entire call request user data to the server RTI-SUI by issuing an RTI-CALL-TASK indication.
2. If cancel call requests are pending, the server RTI-PM issues RTI-CANCEL-CALL indications. The server RTI-PM issues one RTI-CANCEL-CALL indication for every RPC-REMOTE-ALERT indication it has received while the request was in progress (see **RPC-REMOTE-ALERT indication** on page 117).

RPC-RESPONSE indication

This action indicates that all or part of the call result data has been received.

If the Last-Frag parameter is absent, the RTI-PM saves the call result data received on this indication in the segmentation storage.

If the Last-Frag parameter is present, the only or last segment has been received and the client RTI-PM passes the entire call result data to the RTI-SUI by issuing an RTI-CALL-RESULT indication.

RPC-FAULT indication

The RTI-PM issues an RTI-CALL-FAILURE indication. The Reason parameter of the RPC-FAULT indication is mapped onto the Reason parameter of the RTI-CALL-FAILURE indication.

RPC-ORPHANED indication

This indication is received by the server RTI-PM.

The RTI-PM performs the following actions:

1. It issues an RPC-SHUTDOWN request.
2. It reinitialises the segmentation storage.
3. If a call is in progress, it remembers the call has been orphaned.

RPC-REMOTE-ALERT indication

This indication is received by the server RTI-PM.

The RTI-PM performs the following actions:

1. If a request is in progress, it increments the number of the pending cancel call requests.
2. If a call is in progress, it issues an RTI-CANCEL-CALL indication.
3. If a response is in progress, it takes no action

RPC-DONE indication

When received by the client RTI-PM, the client RTI-PM takes the following actions:

1. If segmentation is required, it issues an RPC-REQUEST request without the Last-Frag parameter to transmit the next segment of the call data.
2. If segmentation is not required, it issues an RPC-REQUEST request with the Last-Frag parameter to transmit the last segment of the call data.

When received by the server RTI-PM, the server RTI-PM takes the following actions:

1. If segmentation is required, issues an RPC-RESPONSE request without the Last-Frag parameter.
2. If segmentation is not required, it issues an RPC-RESPONSE request with the Last-Frag parameter present.

RPC-SHUTDOWN indication

This procedure applies to the non-transactional context only.

The client RTI-PM performs the following actions:

1. If a request is in progress or a call is in progress:
   a. It issues an RPC-NO-CONN request.
   b. It issues an RTI-CALL-FAILURE indication with the Reason parameter set to RPC-INSUFFICIENT-RESOURCES.
2. It issues a TP-END-DIALOGUE request.
3. It releases the context and issues an RTI-RELEASE-CONTEXT indication.

8.5.6 TP Indication and Confirmation Procedures

The procedures for TP indications and confirmations are as follows:

TP-BEGIN-DIALOGUE confirmation

This confirmation is received by the client RTI-PM while a call is outstanding.[13]

13. Since dialogue establishment is initiated on the first call, the TP-BEGIN-DIALOGUE confirmation can only be received while a call is outstanding. For transactional contexts, the above condition also guarantees that this confirmation is received prior to transaction termination. Therefore, the Rollback parameter is always set to false.

The RTI-PM performs the following actions:

1. It issues an RTI-CALL-FAILURE indication.
2. For the non-transactional context, it releases the context and issues an RTI-RELEASE-CONTEXT indication.
3. It issues an RPC-NO-CONN request.

TP-END-DIALOGUE indication

This indication is received by the server RTI-PM for a dialogue (transactional or non-transactional).

The context is released.

TP-HEURISTIC-REPORT indication

This procedure applies to the transactional context only.

The client RTI-PM is informed of the actual or possible heuristic inconsistency within the subordinate subtree when it receives a TP-HEURISTIC-REPORT indication.

The client RTI-PM issues an RTI-HEURISTIC-REPORT indication.

TP-U-ABORT indication

For the transactional context, if the server RTI-SUI has issued an RTI-TRANS-DONE request, the server RTI-PM issues a TP-DONE request; otherwise, no action is taken.[14]

For the non-transactional context, the client RTI-PM performs the following actions:

1. If a request or a call is in progress, it issues an RTI-CALL-FAILURE indication with the Reason parameter set to PROTOCOL-MACHINE-FAILURE.
2. It releases the context and issues an RTI-RELEASE-CONTEXT indication.
3. It issues an RPC-NO-CONN request.

For the non-transactional context, the server RTI-PM performs the following actions:

1. If a call is in progress, it remembers the dialogue is aborted.
2. If a request, a call, or a response is in progress, it issues an RPC-NO-CONN request.

TP-P-ABORT indication

For the transactional context:

- If the Rollback parameter is set to "true":
  - If a request or a call is in progress, the client RTI-PM performs the following actions:
    1. It issues an RTI-CALL-FAILURE indication.
    2. It issues an RPC-NO-CONN request.

14. The above restriction is to ensure that the first TP-DONE request following a rollback initiating service primitive is issued to TPPM only after the bound data handled by the requester RTI-SUI is placed in the initial state.

— If a request, a call, or a response is in progress, the server RTI-PM issues an RPC-NO-CONN request.

— The client or server RTI-PM issues an RTI-ROLLBACK-TRANS indication.

- If the Rollback parameter is set to "false" and the second phase of commitment has begun, the RTI-PM issues a TP-DONE request.[15]

For the non-transactional context, the client RTI-PM performs the following actions:

1. If a request or a call is in progress, it issues an RTI-CALL-FAILURE indication.
2. It releases the context and issues an RTI-RELEASE-CONTEXT indication.
3. It issues an RPC-NO-CONN request.

For the non-transactional context, the server RTI-PM performs the following actions:

1. If a call is in progress, it records that the dialogue is aborted.
2. If a request, a call, or a response is in progress, it issues an RPC-NO-CONN request.

TP-DEFERRED-END-DIALOGUE indication

This procedure applies to the transactional context only.

This indication is received by the server RTI-PM.

No action is taken.

TP-BEGIN-TRANSACTION indication

This procedure applies only to non-transactional contexts with transactions enabled.

This indication is received by the server RTI-PM.

If the RTI-PM accepts the request, the context is made transactional.

If the RTI-PM rejects the request, it issues the RTI-CALL-FAILURE request with the Reason parameter set to CONTEXT-TYPE-NOT-SUPPORTED (a transactional RPC was issued, but the server RTI-PM does not support transactional operations).

TP-PREPARE indication

This procedure applies to the transactional context only.

A TP-PREPARE indication is received by an intermediate or leaf node only.

The server RTI-PM issues an RTI-PREPARE-TRANS indication.

15. For the transactional context, the TP-P-ABORT indication without the Rollback parameter can be received only after completion of phase one of the two-phase commit protocol. The absence of the Rollback parameter signals the start of the recovery following a communication failure.

TP-COMMIT indication

This procedure applies to the transactional context only.

The client or server RTI-PM issues an RTI-COMMIT-TRANS indication.

TP-COMMIT-COMPLETE indication

This procedure applies to the transactional context only.

The client or server RTI-PM issues an RTI-TRANS-COMPLETE indication.

All transactional dialogues (without any active context handles) are terminated and the client or server RTI-PM releases all related contexts.

TP-READY indication

This procedure applies to the transactional context only.

If all transaction branches have reported ready, the RTI-PM generates an RTI-TRANS-READY indication to the RTI-SUI.

TP-ROLLBACK indication

This procedure applies to the transactional context only.

This indication may be received by the client or the server RTI-PM at any time before the end of phase one of the two-phase commit.

The RTI-PM performs the following actions:

1. On the client side, if a request or a call is in progress:

   a. It issues an RTI-CALL-FAILURE indication with the Reason parameter set to ROLLBACK-IN-PROGRESS.

   b. It issues an RPC-NO-CONN request.

   On the server side, if a request, a call, or a response is in progress, it issues an RPC-NO-CONN request.

2. It issues an RTI-ROLLBACK-TRANS indication.

TP-ROLLBACK-COMPLETE indication

This procedure applies to the transactional context only.

The client or server RTI-PM issues an RTI-TRANS-COMPLETE indication.

All transactional dialogues (without any active context handles) are terminated and the client or server RTI-PM releases the related transactional contexts.

8.5.7 Internal Events

The procedures for internal events generated by the RTI-PM are as follows.

Internal-Call-Error

An Internal-Call-Error is an error detected by the RTI-PM that results in a call failure. The scope of the error is limited to the call that is in progress.

If the client RTI-PM has a request or call in progress it performs the following actions:

1. It issues an RPC-ORPHANED request.
2. It issues an RTI-CALL-FAILURE indication with the Reason parameter set to PROTOCOL-MACHINE-FAILURE.

The server RTI-PM performs the following actions:

1. If a request is in progress, it issues an RPC-FAULT request with the DNE parameter. The Reason code is set to reflect the cause of the call error. The Reason code may be any defined by the RPC-FAULT request service primitive.
2. If a call is in progress or a response is in progress, it issues an RPC-FAULT request without the DNE parameter. The Reason code is set to reflect the cause of the call error. The Reason code may be any defined by the RPC-FAULT request service primitive.
3. For the non-transactional context, it issues an RPC-SHUTDOWN request. If a call or a response is in progress, it considers the call being orphaned and remembers this fact.

Internal-Fatal-Error

An Internal-Fatal-Error is an error detected by the RTI-PM that results in the immediate termination of the dialogue.

For the non-transactional context the client RTI-PM performs the following actions:

1. It issues a TP-U-ABORT request.
2. If a request or a call is in progress:
    a. It issues an RPC-NO-CONN request.
    b. It issues an RTI-CALL-FAILURE indication with the Reason parameter set to PROTOCOL-MACHINE-FAILURE.
3. It releases the context and issues an RTI-RELEASE-CONTEXT indication.

For the transactional context the client RTI-PM performs the following actions:

1. If a request or a call is in progress, it issues an RTI-CALL-FAILURE indication with the Reason parameter set to PROTOCOL-MACHINE-FAILURE.
2. It issues an RTI-ROLLBACK-TRANS indication
3. It issues a TP-ROLLBACK request.

For the non-transactional context the server RTI-PM performs the following actions:

1. It issues a TP-U-ABORT request.
2. It issues an RPC-NO-CONN request.

For the transactional context the server RTI-PM performs the following actions:

- If the transaction could still be rolled back, that is before phase 2 of the two-phase commit protocol:
  1. It issues a TP-ROLLBACK request.
  2. It issues an RTI-ROLLBACK-TRANS indication.[16]
- If a request, a call or a response is in progress, it issues an RPC-NO-CONN request.

8.6 RTI-APDU Concatenation Rules

RTI-APDUs may be concatenated according to the TP concatenation rules as stated in Clause 11.3 of the OSI TP Protocol standard. In this clause, the term UASE-APDU represents the RTI-APDUs. No additional rules are required.

16. In the case of the first call on a dialogue, the above RTI-ROLLBACK-TRANS indication is issued only if a call or a response is in progress, or if the transaction is in phase one of the two phase commit protocol.

8.7 Sequencing Rules and State Tables

This section contains the state tables for the RTI-PM. All sequencing rules are implicit in the state tables. In order to simplify and enhance readability, the client and server state tables are separated into transactional and non-transactional cases. State transitions for non-transactional contexts are described by the non-transactional state tables, and state transitions for transactional contexts are described by the transactional state tables. Transitions occur between these state tables as context changes from transactional to non-transactional and vice-versa.

When a new context is created, its initial state is I (the Idle State) in the RTI Protocol Client Non-transactional State Table. The initial state for a server is I (the Idle State) in the RTI Protocol Server Non-transactional State Table.

8.7.1 State Table Conventions

In each state table:

- Each column (except the two left-most columns) represents a state.
- Each row (except the first) represents an event.
- Each non-blank cell of the table represents a state transition.

State transitions are controlled by two factors, the incoming event triggering the transition and zero or more precondition variables.

Incoming events are listed in the left-most column of the state table (labelled **Events**).

All precondition variables with the exception of *pcp* are booleans and can therefore take a value of TRUE or FALSE. The precondition variable *pcp* is an integer and can thus take any integer value. If the precondition variable *pcp* is listed in the state table followed by the expression =0 (equal to zero), the value of *pcp* must be zero for the transition to occur. Likewise if the precondition variable *pcp* is listed in the state table followed by the expression ≠0 (not equal to 0), the value of *pcp* must not be zero for the transition to occur.

If a precondition variable is listed in the state table prefixed by ¬ (logical NOT), the value of that variable must be FALSE for the transition predicated by that variable to occur. If a precondition variable has no prefix, the value of that variable must be TRUE for the transition predicated by that variable to occur.

Precondition variables are listed in the second column of the table (labelled **Preconditions**). All precondition variables must evaluate as specified for any transition in that row of the state table to occur. Precondition variables are evaluated from top to bottom with the first precondition variable (or group of precondition variables) that evaluate being the ones to qualify the event. Precondition variables are of the form *px*, where *x* is the two letter abbreviated name of the variable.

Each cell of the state table may contain a set of actions or outgoing events (or both) to be performed before the transition represented by the cell is made. Actions differ from outgoing events in the following way; the results of action routines are not visible outside the RTI-PM, whereas the results of outgoing events are always visible outside the RTI-PM. This distinction is made for clarity and ease of reading. Actions are of the form A*n* where *n* is an integer. Outgoing events are of the form *x* where *x* is a three or four letter abbreviated name of the outgoing event.

Lastly, each cell of the state table that represents a valid state transition contains the name of the resulting state after the transition.

If no valid state transition is listed in the cell at the intersection of a given event and a given state, it is illegal for that event to occur while in that state.

Variables

pab Dialogue has been aborted

This variable, when TRUE, indicates that a dialogue has been aborted. It is initialised and modified by action routines.

pcc At least one call has been started

This variable, when TRUE, indicates that at least one call has been started. It is initialised and modified by action routines.

pch Context handles active

This variable, when TRUE, indicates that the dialogue has one or more active context handles associated with it. It is initialised to FALSE at the creation of the RTI-PM. It is set by the RTI-PM when an RPC_RESPONSE.req is issued or an RPC-RESPONSE.ind is received. It is set TRUE when the first context handle is created and FALSE when the last context handle is destroyed. It is used to control whether or not a dialogue should be terminated when the transaction completes (commits or aborts).

pcp Cancel pending

This variable, unlike all other variables, is an integer. It is initialised to zero by an action routine and incremented by an action routine each time an RPC-REMOTE-ALERT.ind is received.

pra Ready All

This variable, when TRUE, indicates that a TP-READY.ind has been received and that all other transaction branches have been previously indicated as ready.

psd Segment data

This variable, when TRUE, indicates that the data supplied on an RTI-CALL-TASK.req must be segmented. It is set by the RTI-PM each time an RTI-CALL-TASK.req or RPC-DONE.ind is received.

prt Root of the transaction tree

This variable, when TRUE, indicates that this is the root of the transaction tree. It is initialised at the creation of the RTI-PM and is never modified.

plf Last Fragment

This variable, when TRUE, indicates that Last-Frag parameter is present. It is set by the RTI-PM each time an RPC-REQUEST.ind is received.

ptc Transactional call

This variable, when TRUE, indicates that the call is a transactional RPC (**transaction_mandatory** or **transaction_optional** within the scope of a global transaction). It is set by the RTI-PM each time an RTI-CALL-TASK.req is received. It is used to control whether a non-transactional dialogue should be included in the current transaction.

pte Transaction Enabled

This variable, when TRUE, indicates that transactions are enabled. It is set when context is established.

8.7.2 Keys to State Table Abbreviations

The following tables:

- Table 8-24
- Table 8-25
- Table 8-26 on page 126
- Table 8-27 on page 126
- Table 8-28 on page 127
- Table 8-29 on page 127
- Table 8-30 on page 128
- Table 8-31 on page 128

are keys to the RTI Protocol State Tables:

- Table 8-32 on page 129
- Table 8-33 on page 131
- Table 8-34 on page 133
- Table 8-35 on page 134.

Table 8-24 Abbreviations for Client Non-transactional States

Abbrev.	Meaning
I	Idle
CE	Context Established
DE	Dialogue Established
RIP	Request in Progress
CIP	Call in Progress

Table 8-25 Abbreviations for Client Transactional States

Abbrev.	Meaning
DE	Dialogue Established
RIP	Request in Progress
CIP	Call in Progress
SP1	Start of Phase 1
EP1	End of Phase 1
SP2	Start of Phase 2
EP2	End of Phase 2
RBP	Rollback Pending
RBC	Rollback Complete

Table 8-26 Abbreviations for Server Non-transactional States

Abbrev.	Meaning
I	Idle
DE	Dialogue Established
RIP	Request in Progress
CIP	Call in Progress
RSP	Response in Progress
ORP	Orphaned Call

Table 8-27 Abbreviations for Server Transactional States

Abbrev.	Meaning
DE	Dialogue Established
RIP	Request in Progress
CIP	Call in Progress
RSP	Response in Progress
ORP	Orphaned Call
SP1	Start of Phase 1
EP1	End of Phase 1
SP2	Start of Phase 2
EP2	End of Phase 2
RBP	Rollback Pending
RBC	Rollback Complete

Table 8-28 RTI Protocol Client Outgoing Events

Abbrev.	Meaning
rabd	DC-BEGIN-DIALOGUE.req
rrtr	RPC-REQUEST.req without Last-Frag
rrlt	RPC-REQUEST.req with Last-Frag
rrle	RPC-ORPHANED.req
rrla	RPC-REMOTE-ALERT.req
rrnc	RPC-NO-CONN.req
rtbt	TP-BEGIN-TRANSACTION.req
rtde	TP-DEFERRED-END-DIALOGUE.req
rtrb	TP-ROLLBACK.req
rtct	TP-COMMIT.req
rtdn	TP-DONE.req
rted	TP-END-DIALOGUE.req
rtpr	TP-PREPARE.req
rtab	TP-U-ABORT.req
icf	RTI-CALL-FAILURE.ind
icr	RTI-CALL-RESULT.ind
itr	RTI-TRANS-READY.ind
ict	RTI-COMMIT-TRANS.ind
ihr	RTI-HEURISTIC-REPORT.ind
ite	RTI-TRANS-COMPLETE.ind
irc	RTI-RELEASE-CONTEXT.ind
irt	RTI-ROLLBACK-TRANS.ind

Table 8-29 RTI Protocol Server Outgoing Events

Abbrev.	Meaning
rarj	DC-REJECT-DIALOGUE.req
rrrp	RPC-RESPONSE.req without Last-Frag
rrfd	RPC-FAULT.req with DNE
rrfe	RPC-FAULT.req without DNE
rrnc	RPC-NO-CONN.req
rrlr	RPC-RESPONSE.req with Last-Frag
rtdn	TP-DONE.req
rtrb	TP-ROLLBACK.req
rtcc	TP-COMMIT.req
rtab	TP-U-ABORT.req
ica	RTI-CALL-TASK.ind
icc	RTI-CANCEL-CALL.ind
ipt	RTI-PREPARE-TRANS.ind
irt	RTI-ROLLBACK-TRANS.ind
ict	RTI-COMMIT-TRANS.ind
ite	RTI-TRANS-COMPLETE.ind

Table 8-30 RTI Protocol Preconditions

Abbrev.	Meaning
pab	The dialogue has been aborted
pcc	At least one call has been started
pch	The dialogue has one or more active context handles
pcp	Cancel pending
psd	User argument data requires segmentation, that is, the argument data is larger than the segment size
prt	Client is the root of the transaction
plf	Last fragment parameter is present
ptc	The call is transactional
pte	Transactions are enabled

Table 8-31 RTI Protocol Actions

Abbrev.	Meaning
A0	Set pcp to 0
A1	Set pab, pcc to false
A2	Assemble user data
A3	Set pab to true
A4	Add 1 to pcp
A5	Set pcc to true

8.7.3 RTI Protocol State Tables

The following state tables describe the RTI protocol .

Table 8-32 RTI Protocol Client Non-transactional State Table

Events	Precon-ditions	I (Idle)	CE	DE	RIP	CIP
RTI-ESTABLISH-CONTEXT.req		CE				
RTI-CALL-TASK.req	¬ptc ¬psd		rabd rrlt CIP	rrlt CIP		
	¬ptc psd		rabd rrtr RIP	rrtr RIP		
	ptc pte ¬psd		rabd rtbt rrlt CIP †	rtbt rrlt CIP †		
	ptc pte psd		rabd rtbt rrtr RIP †	rtbt rrtr RIP †		
RTI-CANCEL-CALL.req					rrla RIP	rrla CIP
RPC-RESPONSE.ind	¬plf					A2 CIP
	plf					icr DE
RPC-FAULT.ind					icf DE	icf DE
RPC-DONE.ind	¬psd				rrlt CIP	
	psd				rrtr RIP	
TP-BEGIN-DIALOGUE.cnf					icf irc rrnc I	icf irc rrnc I
DC-REJECT-DIALOGUE.ind					icf irc rrnc I	icf irc rrnc I

Events	Precon-ditions	I (Idle)	CE	DE	RIP	CIP
TP-U-ABORT.ind				irc rrnc I	icf irc rrnc I	icf irc rrnc I
TP-P-ABORT.ind				irc rrnc I	icf irc rrnc I	icf irc rrnc I
RTI-RELEASE-CONTEXT.req			I	rted I		
RPC-SHUTDOWN.ind				irc rted I	rrnc rted icf irc I	rrnc rted icf irc I
Internal-Call-Error					rrle icf DE	rrle icf DE
Internal-Fatal-Error			irc I	rtab irc I	rtab rrnc icf irc I	rtab rrnc icf irc I

Notes:

† This is the initial state table for all client RTI-PMs. Initiation of a transactional RTI-CALL-TASK.req (ptc) results in the execution of a TP-BEGIN-TRANSACTION.req (rtbt), and a switch to the RTI Protocol Client Transactional State Table (see Table 8-33 on page 131).

Table 8-33 RTI Protocol Client Transactional State Table

Events	Precon-ditions	DE	RIP	CIP	SP1	EP1	SP2	EP2	RBP	RBC
RTI-CALL-TASK.req	¬psd	rrlt CIP								
	psd	rrtr RIP								
RTI-CANCEL-CALL.req			rrla RIP	rrla CIP						
RTI-ROLLBACK-TRANS.req		rtrb RBP			rtrb RBP					
RTI-END-TRANS.req	prt pch	rtpr SP1								
	prt ¬pch	rtde rtpr SP1								
RTI-COMMIT-TRANS.req	prt					rtct SP2				
RTI-TRANS-DONE.req	pch						rtdn EP2		rtdn RBC	
	¬pch						rtdn EP2		rtab rtdn RBC	
RPC-RESPONSE.ind	¬plf			A2 CIP						
	plf			icr DE						
RPC-FAULT.ind			icf DE	icf DE						
RPC-DONE.ind	¬psd		rrlt CIP							
	psd		rrtr RIP							
TP-BEGIN-DIALOGUE.cnf Result=rejected(provider)			icf rrnc I ‡	icf rrnc I ‡						
DC-REJECT-DIALOGUE.ind			icf rrnc rtrb I †	icf rrnc rtrb I †						
TP-HEURISTIC-REPORT.ind							ihr SP2	ihr EP2	ihr RBP	ihr RBC
TP-P-ABORT.ind Rollback="false"						EP1	SP2	rtdn EP2	RBP	
TP-P-ABORT.ind Rollback="true"		irt RBP	icf irt rrnc RBP	icf irt rrnc RBP	irt RBP	irt RBP				

Events	Precon-ditions	DE	RIP	CIP	SP1	EP1	SP2	EP2	RBP	RBC
TP-COMMIT.ind							ict SP2			
TP-COMMIT-COMPLETE.ind	pch							ite DE †		
	¬pch							ite I †		
TP-ROLLBACK.ind		irt RBP	icf rrnc irt RBP	icf rrnc irt RBP	irt RBP	irt RBP				
TP-ROLLBACK-COMPLETE.ind	pch									ite DE †
	¬pch									ite rted I †
TP-READY.ind	¬pra				SP1					
	pra				itr EP1					
Internal-Call-Error			icf rrle DE	icf rrle DE						
Internal-Fatal-Error		rtrb irt RBP	icf rtrb irt RBP	icf rtrb irt RBP	rtrb irt RBP					

Notes:

† One or more calls can occur on a dialogue while it is joined to a transaction. Between calls, the state is DE (Dialogue Established). Eventually, the transaction is either committed or rolled back. In either case, this results in a shift back to the RTI Protocol Client Non-Transactional State Table (see Table 8-32 on page 129). The new state depends on whether or not there are still active context handles. If so (pch), the new state is DE (Dialogue Established). Otherwise (¬pch), the dialogue is terminated with the transaction, so the new state is I (Idle). These state changes occur for TP-COMMIT-COMPLETE.ind and TP-ROLLBACK-COMPLETE.ind. For TP-COMMIT-COMPLETE.ind, whether or not the dialogue is terminated at transaction completion is effected by making TP-DEFERRED-END-DIALOGUE.req (rtde) conditional on no context handles being active (¬pch) on the RTI-END-TRANS.req. Because the dialogue is not terminated by TP-DEFERRED-END-DIALOGUE if the transaction is rolled back, for TP-ROLLBACK-COMPLETE.ind, TP-END-DIALOGUE.req (rted) is issued if there are no active context handles (¬pch), to terminate the dialogue.

‡ If the initial call on a dialogue is transactional and the dialogue is rejected (TP-BEGIN-DIALOGUE.cnf(result = rejected), or DC-REJECT-DIALOGUE.ind), the state transition is to I (Idle) in the RTI Protocol Client Non-Transactional State Table (see Table 8-32 on page 129).

Table 8-34 RTI Protocol Server Non-transactional State Table

Events	Precon-ditions	I (Idle)	DE	RIP	CIP	RSP	ORP
DC-BEGIN-DIALOGUE.ind		A1 DE -or- rarj I					
RTI-CALL-RESULT.req	¬psd ¬pab				rrlr DE		DE
	psd ¬pab				rrrp RSP		DE
	pab				I		I
RPC-REQUEST.ind	¬plf		A0 A2 RIP	A2 RIP			
	plf pcp=0		ica CIP	ica CIP			
	plf pcp≠0		ica icc CIP	ica icc CIP			
RPC-DONE.ind	¬psd					rrlr DE	
	psd					rrrp RSP	
RPC-REMOTE-ALERT.ind				A4 RIP	icc CIP	RSP	
RPC-ORPHANED.ind				DE	ORP	DE	
TP-BEGIN-TRANSACTION.ind	pte		DE †				
TP-U-ABORT.ind			I	rrnc I	rrnc A3 CIP	rrnc I	A3 ORP
TP-P-ABORT.ind			I	rrnc I	rrnc A3 CIP	rrnc I	A3 ORP
TP-END-DIALOGUE.ind			I				A3 ORP
Internal-Call-Error				rrfd DE	rrfe ORP	rrfe DE	
Internal-Fatal-Error			rtab I	rtab rrnc I	rtab rrnc I	rtab rrnc I	rtab I

Notes:

† This is the initial state table for all server RTI-PMs. Receipt of a TP-BEGIN-TRANSACTION.ind results in a switch to the RTI Protocol Server Transactional State Table (see Table 8-35 on page 134).

Table 8-35 RTI Protocol Server Transactional State Table

Events	Precon-ditions	DE	RIP	CIP	RSP	ORP	SP1	EP1	SP2	EP2	RBP	RBC
RTI-CALL-RESULT.req	¬psd			rrlr DE		DE						
	psd			rrrp RSP		DE						
RTI-ROLLBACK-TRANS.req	pcc		rrfd rtrb RBP		rrfe rtrb RBP		rtrb RBP					
RTI-TRANS-READY.req							rtcc EP1					
RTI-TRANS-DONE.req									rtdn EP2		rtdn RBC	
RPC-REQUEST.ind	¬plf	A0 A2 RIP	A2 RIP									
	plf pcp=0	A5 ica CIP	A5 ica CIP									
	plf pcp≠0	A5 ica icc CIP	A5 ica icc CIP									
RPC-DONE.ind	¬psd				rrlr DE							
	psd				rrrp RSP							
RPC-ORPHANED.ind			DE	ORP	DE							
RPC-REMOTE-ALERT.ind			A4 RIP	icc CIP	RSP							
TP-U-ABORT.ind											RBP	rtdn RBC
TP-P-ABORT.ind Rollback="false"								EP1	SP2	rtdn EP2	RBP	
TP-P-ABORT.ind Rollback="true"		irt RBP	irt rrnc RBP	irt rrnc RBP	irt rrnc RBP	irt RBP	irt RBP	irt RBP				
TP-PREPARE.ind		ipt SP1										
TP-COMMIT.ind								ict SP2				
TP-COMMIT-COMPLETE.ind	pch									ite DE †		
	¬pch									ite I †		
TP-ROLLBACK.ind		irt RBP	irt rrnc RBP	irt rrnc RBP	irt rrnc RBP	irt RBP	irt RBP	irt RBP				
TP-ROLLBACK-COMPLETE.ind	pch											ite DE †
	¬pch											ite DE

Events	Precon-ditions	DE	RIP	CIP	RSP	ORP	SP1	EP1	SP2	EP2	RBP	RBC
TP-DEFERRED-END-DIALOGUE.ind		DE										
TP-END-DIALOGUE.ind		I †										
Internal-Call-Error			rrfd DE	rrfe DE	rrfe DE	DE						
Internal-Fatal-Error	¬pcc	I ‡	rrnc I ‡	irt rtrb rrnc RBP	irt rtrb rrnc RBP	irt rtrb RBP	irt rtrb RBP					
	pcc	irt rtrb RBP	irt rtrb rrnc RBP	irt rtrb rrnc RBP	irt rtrb rrnc RBP	irt rtrb RBP	irt rtrbb RBP					

Notes:

† One or more calls can occur on a dialogue while it is joined to a transaction. Between calls, the state is DE (Dialogue Established). Eventually, the transaction is either committed or rolled back. In either case, this results in a shift back to the RTI Protocol Server Non-Transactional State Table (see Table 8-34 on page 133). The new state depends on whether or not there are still active context handles. If so (pch), the new state is DE (Dialogue Established). Otherwise (¬pch), the dialogue is terminated with the transaction, so the new state is I (Idle). These state changes occur for TP-COMMIT-COMPLETE.ind and TP-END-DIALOGUE.ind. For TP-COMMIT-COMPLETE.ind, whether or not the dialogue is terminated at transaction completion is effected by TP-DEFERRED-END-DIALOGUE.ind, which is only be received if there are no active context handles(¬pch). Because the dialogue is not terminated by TP-DEFERRED-END-DIALOGUE if the transaction is rolled back, for TP-ROLLBACK-COMPLETE.ind the new state is DE (Dialogue Established) in this state table. Subsequently, if there are no active context handles (¬pch), TP-END-DIALOGUE.ind will be received and the new state is I (Idle) in the RTI Protocol Server Non-Transactional State Table (see Table 8-34 on page 133).

‡ Switch to Table 8-34 on page 133.

Chapter 9

Architectural Constants

This chapter contains known architectural constants.

9.1 RPC Architectural Constants

The following RPC architectural constants are relevant:

- Transfer Syntax: ASN.1/BER, NDR
- MaxFragSize: 10228[17]
- PFC_CONC_MPX: 0
- PFC_MAYBE: 0
- The values for the reason code on the RPC-FAULT request and therefore the status field on the rpc_fault APDU are given in Table 9-1 in hexadecimal form. For a complete listing of constants see Section 12.6, Connection-oriented RPC PDUs, of the X/Open **DCE RPC** specification.

Table 9-1 Values for Fault Reasons

Reason Code	Value
RPC-ACCESS-VIOLATION	0x1C000002
RPC-CANCEL	0x1C00000D
RPC-FLOATING-DIVIDE-BY-ZERO	0x1C000003
RPC-FLOATING-ERROR	0x1C00000F
RPC-FLOATING-OVERFLOW	0x1C000005
RPC-FLOATING-UNDERFLOW	0x1C000004
RPC-INSUFFICIENT-RESOURCES	0x1C010014
RPC-INTEGER-DIVIDE-BY-ZERO	0x1C000001
RPC-INTEGER-OVERFLOW	0x1C000010
RPC-INVALID-OPERATION-NUMBER	0x1C010002
RPC-INVOCATION-FAILURE	0x1C00000C
RPC-MARSHALLING-ERROR	0x1C010017
RPC-PROTOCOL-ERROR	0x1C01000B
RPC-REASON-NOT-SPECIFIED	0x1C000012

17. MaxFragSize is computed from Maximum SSDU size (10240) minus the minimum Presentation Layer header size for large fragments (12).

- The packet type values for the RPC packets used by RTI are given in Table 9-2[18]. These are values for the PTYPE APDU field in decimal form.

Table 9-2 Packet Type Values

Packet Type	Value
rpc_request	0
rpc_response	2
rpc_fault	3
rpc_shutdown	17
rpc_remote_alert	18
rpc_orphaned	19

18. These values will be removed in a future version of this specification when the packet type values have been clearly documented in the X/Open **DCE RPC** specification.

X/Open CAE Specification

Part 3:
TxRPC Communication API Appendices

X/Open Company Ltd.

Appendix A

RPC TxRPC Example

This appendix contains an example using an RPC TxRPC CRM. It consists of the IDL file which describes the TxRPC and the C-language code which implements the client, the server and the manager functions for the TxRPC.

This example is not a complete application, it illustrates the various components necessary to implement a TxRPC application.

A.1 IDL File

```
/*
*
* debitCredit.idl
*
* "Debit/Credit" program for TxRPC
*
*
*/

[
uuid(006690B8-23C6-19EC-A494-C037CF540000),
version(1.0), transaction_mandatory
]

interface debitCredit
{

        void debit(
               [in]   handle_t h,
               [in]   long int account,
               [in]   long int amount
        );

        void credit(
               [in]   handle_t h,
               [in]   long int account,
               [in]   long int amount
        );

}
```

A.2 Common Include Files — <util.h>

```
/*
 * util.h
 *
 * Declarations of utility routine(s) shared by "debitCredit" client
 * and server programs.
 */

#define ERROR_CHECK(status, text) \
       if (status != error_status_ok) error_exit(status, text)

void error_exit( error_status_t status, char *text);

#define TX_ERROR_CHECK(status, text) \
       if (status != TX_OK) fprintf(stderr, "%s: %d\n", text, status)

#define TXRPC_ERROR_CHECK(status, text) \
       if (status != TXRPC_OK) fprintf(stderr, "%s: %d\n", text, status)
```

A.3 Client Side

```
/*
 * debitCredit.c
 *
 * Client of "debitCredit" interface.
 */

#include <stdio.h>

#include "debitCredit.h"
#include "util.h"
#include <tx.h>

int main(int argc, char *argv[])
{
   rpc_ns_handle_t import_context;
   handle_t binding_h;
   error_status_t status;
   idl_char reply[REPLY_SIZE];
   int tx_status;

   if (argc < 2) {
           fprintf(stderr, "usage: debitCredit <CDS pathname>\n");
           exit(1);
   }


   /*
    * Start importing servers using the name specified
    * on the command line.
    */

   rpc_ns_binding_import_begin( rpc_c_ns_syntax_default,
                                (unsigned_char_p_t) argv[1],
                                debitCreditif_v1_0_c_ifspec,
                                NULL,
                                & import_context,
                                & status);
   ERROR_CHECK(status, "Can't begin import");

   /*
    * Import the first server (we could iterate here,
    * but we'll just take the first one).
    */

   rpc_ns_binding_import_next( import_context,
                               & binding_h,
                               & status);
   ERROR_CHECK(status, "Can't import");
```

```
    /*
     * Initialise the (C)RMs
     */

    tx_status = tx_open();
    TX_ERROR_CHECK(tx_status, "Can't open");

    /*
     * Start the transaction
     */

    tx_status = tx_begin();
    TX_ERROR_CHECK(tx_status, "Can't begin");


    /*
     * Make the remote calls.
     */
    debit(binding_h, /* from account */ 100, /* amount */ 200);

    credit(binding_h, /* to account */  101, /* amount */ 200);


    /*
     * End the transaction
     */

    tx_status = tx_commit();
    TX_ERROR_CHECK(tx_status, "Can't commit");


    /*
     * Finalise the (C)RMs
     */

    tx_status = tx_close();
    TX_ERROR_CHECK(tx_status, "Can't close");
}
```

A.4 Server Side

```
/*
 * debitCreditServer.c
 *
 * Main program (initialisation) for "debitCredit" server.
 */

#include <stdio.h>

#include "debitCredit.h"
#include "util.h"

int main(int argc, char *argv[])
{
   unsigned32 status;
   rpc_binding_vector_t *binding_vector;
   int tx_status;

   if (argc < 2) {
          fprintf(stderr,
                  "usage: debitCreditServer <CDS pathname>\n");
          exit(1);
   }

   /*
    * Register interface with RPC run-time.
    */

   rpc_server_register_if(debitCreditif_v1_0_s_ifspec,
                          NULL,
                          NULL,
                          & status);
   ERROR_CHECK(status, "Can't register interface");

   /*
    * Use OSI TP protocol sequence.
    */

   rpc_server_use_protseq(ncacn_osi_tp, rpc_c_protseq_max_reqs_default,
                                & status);
   ERROR_CHECK(status, "Can't use protocol sequences");

   /*
    * Get the binding handles generated by the run-time.
    */

   rpc_server_inq_bindings(&binding_vector,
                           &status);
   ERROR_CHECK(status, "Can't get bindings for server");

   /*
    * Register assigned endpoints with endpoint mapper (RPCD).
```

```
 */

rpc_ep_register(debitCreditif_v1_0_s_ifspec,
                binding_vector,
                NULL,
                (unsigned_char_p_t) "debitCredit server version 1.0",
                & status);
ERROR_CHECK(status, "Can't register with endpoint map");

/*
 * Export ourselves to the into the CDS name space.
 */

rpc_ns_binding_export( rpc_c_ns_syntax_default,
                       (unsigned_char_p_t) argv[1],
                       debitCreditif_v1_0_s_ifspec,
                       binding_vector,
                       NULL,
                       & status);
ERROR_CHECK(status, "Can't export to CDS name space");

/*
 * Initialise the (C)RMs
 */

tx_status = tx_open();
TX_ERROR_CHECK(tx_status, "Can't open");

/*
 * Start listening for calls.
 */

printf("Listening ...\n");

rpc_server_listen( rpc_c_listen_max_calls_default,
                   & status);
ERROR_CHECK(status, "Can't start service threads");

/*
 * Unregister from endpoint mapper.
 */

rpc_ep_unregister( debitCreditif_v1_0_s_ifspec,
                   binding_vector,
                   NULL,
                   & status);
ERROR_CHECK(status, "Can't unregister from endpoint map");
```

```
   /*
    * Finalise the (C)RMs
    */

   tx_status = tx_close();
   TX_ERROR_CHECK(tx_status, "Can't close");
}
```

A.5 Manager Functions

```
/*
 * debitCreditManager.c
 *
 * Implementation of "debitCredit" interface.
 */

#include <stdio.h>
#include "debitCredit.h"

void debit( handle_t h,
       idl_long_int account,
       idl_long_int amount)
{
       EXEC SQL /* something appropriate */;

}

void credit( handle_t h,
       idl_long_int account,
       idl_long_int amount)
{
       EXEC SQL /* something appropriate */;

}
```

Appendix B

IDL-only TxRPC Example

This appendix contains the same example as in Appendix A, but using an IDL-only TxRPC CRM. The elements of the example are similar but in this case there is no server main function, only the manager functions.

This example is not a complete application, it illustrates the various components necessary to implement a TxRPC application.

B.1 IDL File

```
/*
 *
 * debitCredit.idl
 *
 * "Debit/Credit" program for TxRPC
 *
 *
 */

[
uuid(006690B8-23C6-19EC-A494-C037CF540000),
version(1.0), transaction_mandatory
]

interface debitCredit
{

        void debit(
               [in]   long int account,
               [in]   long int amount
        );

        void credit(
               [in]   long int account,
               [in]   long int amount
        );

}
```

B.2 Common Include Files — <util.h>

```
/*
 * util.h
 *
 * Declarations of utility routine(s) shared by "debitCredit" client
 * and server programs.
 */

#define ERROR_CHECK(status, text) \
        if (status != error_status_ok) error_exit(status, text)

void error_exit( error_status_t status, char *text);

#define TX_ERROR_CHECK(status, text) \
        if (status != TX_OK) fprintf(stderr, "%s: %d\n", text, status)
```

B.3 Client Side

```
/*
 * debitCredit.c
 *
 * Client of "debitCredit" interface.
 */

#include <stdio.h>

#include "debitCredit.h"
#include "util.h"
#include <tx.h>

int main(int argc, char *argv[])
{
        error_status_t status;
        int tx_status;

        if (argc < 2) {
                fprintf(stderr, "usage: debitCredit <CDS pathname>\n");
                exit(1);
        }

        /*
         * Initialise the (C)RMs
         */

        tx_status = tx_open();
        TX_ERROR_CHECK(tx_status, "Can't open");

        /*
         * Start the transaction
         */

        tx_status = tx_begin();
        TX_ERROR_CHECK(tx_status, "Can't begin");

        /*
         * Make the remote calls.
         */
        debit(/* from account */ 100, /* amount */ 200);

        credit(/* to account */  101, /* amount */ 200);

        /*
         * End the transaction
         */

        tx_status = tx_commit();
        TX_ERROR_CHECK(tx_status, "Can't commit");
```

```
        /*
         * Finalise the (C)RMs
         */

        tx_status = tx_close();
        TX_ERROR_CHECK(tx_status, "Can't close");
}
```

B.4 Manager Functions

```
/*
 * debitCreditManager.c
 *
 * Implementation of "debitCredit" interface.
 */

#include <stdio.h>
#include "debitCredit.h"

void debit( idl_long_int account,
            idl_long_int amount)
{
        EXEC SQL /* something appropriate */;

}

void credit( idl_long_int account,
             idl_long_int amount)
{
        EXEC SQL /* something appropriate */;

}
```

Appendix C

TxRPC API to Protocol Mapping

This appendix describes the mappings between the TxRPC API and the underlying protocol. For reasons of completeness it is necessary to include a discussion of some aspects of the TX API since this API has a direct bearing on the underlying protocol.

The Client protocol procedures, Server protocol procedures, Client table and Server table are designed to be a complete and rigorous discussion of the API to protocol mappings required by a TxRPC CRM. These sections are complementary and should be considered a single body of information.

C.1 Client Events

Listed below are the client TxRPC CRM events that affect the flow of protocol. In general, only events that affect the flow of protocol are discussed. However, some events which help in establishing context or are useful in furthering understanding of the relationship between the protocol and other components of the TxRPC CRM are also discussed.

Each event falls into one of three categories:

- events made visible by actions on the TxRPC CRM API
- events made visible by actions on the TX API
- events made visible by the RTI protocol machine.

The events made visible by the TxRPC CRM API are:

- Call (From AP to the CRM)
- Cancel (From AP to CRM)
- Call-Return (From CRM to AP)
- Exception (From CRM to AP).

The events made visible by the TX API for transactional and non-transactional TxRPC are:[19]

- *tx_close*( ) (From AP to TM)
- *tx_open*( ) (From AP to TM).

In addition, the events made visible by the TX API for transactional TxRPC only are:

- *tx_begin*( ) (From AP to TM)
- *tx_commit*( ) (From AP to TM)
- *tx_rollback*( ) (From AP to TM).

19. These events are defined in the **TX** (Transaction Demarcation) specification.

The events made visible by the RTI protocol machine for transactional and non-transactional TxRPC are:

- RTI-CALL-FAILURE indication (raised by the RTI-PM)
- RTI-CALL-RESULTS indication (raised by the RTI-PM).

In addition, the events made visible by the RTI protocol machine for transactional TxRPC only are:

- RTI-TRANS-READY indication (raised by the RTI-PM)
- RTI-COMMIT-TRANS indication (raised by the RTI-PM)
- RTI-COMMIT-TRANS request (issue to the RTI-PM)
- RTI-END-TRANS request (issue to the RTI-PM)
- RTI-HEURISTIC-REPORT indication (raised by the RTI-PM)
- RTI-ROLLBACK-TRANS indication (raised by the RTI-PM)
- RTI-ROLLBACK-TRANS request (issue to the RTI-PM)
- RTI-TRANS-COMPLETE indication (raised by the RTI-PM)
- RTI-TRANS-DONE request (issue to the RTI-PM).

C.1.1 Call

This event is defined to be the point when control is transferred from the calling procedure in the client to a called procedure (manager function). It is important to distinguish between the point when control is transferred from the calling procedure and the point when the called procedure (manager function) receives control since these are necessarily two separate points in time. The Call event is the former. The Start-Call is the latter.

The Call event marks the start of a TxRPC in the client.

A Call is allowed only after *tx_open*( ). If this is a transactional TxRPC, *tx_begin*( ) must also precede the Call.

The first Call causes an RTI-ESTABLISH-CONTEXT request to be issued. All Call events issue an RTI-CALL-TASK request.

C.1.2 Cancel

This event is defined to be the point when a cancel is requested.

A Cancel can only be issued while a call is in progress. A call is in progress from the point when a Call is issued until the point when an RTI-CALL-FAILURE indication or an RTI-CALL-RESULTS indication is raised. If the Call is transactional, an RTI-ROLLBACK-TRANS indication may also be raised.

A Cancel causes an RTI-CANCEL-CALL request to be issued.

C.1.3 Call-Return

This event is defined to be the point when control is transferred from the called procedure (manager function) in a server to a calling procedure. It is important to distinguish between the point when control is transferred from the called procedure (manager function) and the point when the calling procedure regains control since these are necessarily two separate points in time. The Return event is the former. The Call-Return event is the latter.

A Return marks the end of a TxRPC in the server. This is the normal return from a TxRPC.

A Return is allowed only while a call is in progress at the server. A call is in progress at the server from the point when an RTI-CALL-TASK indication is raised until the point when a Return or Unhandled-Exception is issued.

A Return causes an RTI-CALL-RESULTS request to be issued. If *tx_rollback*( ) has been called the Transaction Context is set to indicate this. See Section 2.8 on page 15.

C.1.4 Unhandled-Exception

This event is defined to be the point when control is transferred from the called procedure (manager function) in a server to a calling procedure via an exception mechanism. It is important to distinguish between the point when control is transferred from the called procedure (manager function) and the point when the calling procedure regains control since these are necessarily two separate points in time. The Unhandled-Exception event is the former. The Exception event is the latter.

An Unhandled-Exception marks the end of a TxRPC in the server. This is an abnormal return from a TxRPC.

The Unhandled-Exception event is allowed only while a call is in progress at the server. A call is in progress at the server from the point when an RTI-CALL-TASK indication is raised until the point when a Return or Unhandled-Exception is issued.

An Unhandled-Exception causes an Internal-Call-Error to be raised.

C.1.5 tx_close( )

The *tx_close*( ) event does not cause protocol to flow. It does cause an important state transition.

C.1.6 tx_open( )

The *tx_open*( ) event must be issued prior to any other events.

The *tx_open*( ) event does not cause protocol to flow. It does cause an important state transition.

C.1.7 RTI-CALL-FAILURE indication

An RTI-CALL-FAILURE indication is raised only when a TxRPC is in progress. A call is in progress from the point when a Call is issued until the point when an RTI-CALL-FAILURE indication, or an RTI-CALL-RESULTS indication is raised. If the Call is transactional, an RTI-ROLLBACK-TRANS indication may also be raised.

An RTI-CALL-FAILURE indication terminates an outstanding TxRPC by raising an exception. If the default exception mechanism is overridden the appropriate parameter is populated and control is returned to the calling procedure.

If the TxRPC is non-transactional, the Reason parameter may indicate that the context has been destroyed, in which case the next Call event re-establishes context.

C.1.8 RTI-CALL-RESULTS indication

An RTI-CALL-RESULTS indication is raised only when a TxRPC is in progress. A call is in progress from the point when a Call is issued until the point when an RTI-CALL-FAILURE indication, or an RTI-CALL-RESULTS indication is raised. If the Call is transactional, then an RTI-ROLLBACK-TRANS indication may also be raised.

An RTI-CALL-RESULTS indication terminates an outstanding TxRPC by raising the Call-Return event. This returns control to the calling procedure. It is important to distinguish between the point when control is transferred from the called procedure (manager function) and the point when the calling procedure regains control since these are necessarily two separate points in time. The Call-Return event is the latter. The Return event is the former.

This is the normal return from a TxRPC.

C.1.9 tx_begin()

The *tx_begin*( ) event is allowed only after *tx_open*( ).

The participant which issues *tx_begin*( ) is the root of the transaction. The root is the coordinator of the transaction.

The *tx_begin*( ) event does not cause protocol to flow. It does cause an important state transition.

C.1.10 tx_commit()

The *tx_commit*( ) event is allowed only after *tx_begin*( ).

The *tx_commit*( ) event is allowed only from the root of a transaction. The root is the coordinator of the transaction.

If one or more TxRPCs have been called then an RTI-END-TRANS request is issued.

If one or more TxRPCs have been called and the local participant is able successfully to place all bound data in the ready state, an RTI-TRANS-READY request is issued. The local participant must be in the READY state[20] when this request is issued.

If one or more TxRPCs have been called and the local node is unable successfully to place all bound data in the ready state, an RTI-ROLLBACK-TRANS request is issued. When the local participant has returned all bound data to the initial state an RTI-TRANS-DONE request is issued.

If no TxRPCs have been called, this event does not cause protocol to flow.

20. This state is defined by the OSI TP Model, Service and Protocol standards.

C.1.11 tx_rollback()

The *tx_rollback*( ) event is allowed only after *tx_begin*( ).

An intermediate or leaf node becomes a client when it issues a TxRPC. This event (issuing a TxRPC) places the client in a state where *tx_rollback*( ) is disallowed. The client remains in this state until either an RTI-CALL-RESULTS indication, an RTI-CALL-FAILURE indication or an RTI-ROLLBACK-TRANS indication causes a state transition. These events mark the completion of a TxRPC and thus the point when an intermediate or leaf node ceases to be a client. Consequently, a client must be the root to issue *tx_rollback*( ).

Servers may also issue *tx_rollback*( ). See Section 2.8 on page 15 for a discussion of this.

If at least one TxRPC has been called an RTI-ROLLBACK-TRANS request is issued. After the local participant has released bound data an RTI-TRANS-DONE request is issued.

If no TxRPCs have been made then this event does not cause protocol to flow.

C.1.12 RTI-TRANS-READY indication

At least one TxRPC must have been raised previously, the client RTI-SUI must have previously issued an RTI-END-TRANS request, and the subordinate transaction branches must all have reported ready for the RTI-TRANS-READY indication to be raised.

After the controlling TM has decided the final status of the transaction, the RTI-SUI may issue either an RTI-COMMIT-TRANS request or an RTI-ROLLBACK-TRANS request.

C.1.13 RTI-COMMIT-TRANS indication

At least one TxRPC must have been made in order for the RTI-COMMIT-TRANS indication to be raised.

An RTI-COMMIT-TRANS indication is raised when the outcome of a transaction is commitment. After the local participant has placed bound data in the final state an RTI-TRANS-DONE request is issued.

If the *commit_return* characteristic[21] has been set to TX_COMMIT_DECISION_LOGGED, control is returned to the calling procedure. Otherwise, the calling procedure remains blocked pending completion of the transaction.

C.1.14 RTI-HEURISTIC-REPORT indication

At least one TxRPC must have been made in order for the RTI-HEURISTIC-REPORT indication to be raised.

An RTI-HEURISTIC-REPORT indication is raised when either a heuristic mix or a heuristic hazard has been detected in the the subordinate transaction subtree.

Policies regarding heuristic outcomes are beyond the scope of this specification.

21. This characteristic is defined in the **TX** (Transaction Demarcation) specification.

C.1.15 RTI-ROLLBACK-TRANS indication

At least one TxRPC must have been made in order for the RTI-ROLLBACK-TRANS indication to be raised.

An RTI-ROLLBACK-TRANS indication is raised when the current transaction is being rolled back. After all bound data is released in the initial state a RTI-TRANS-DONE request is issued.

C.1.16 RTI-TRANS-COMPLETE indication

At least one TxRPC must have been made in order for the RTI-TRANS-COMPLETE indication to be raised.

An RTI-TRANS-COMPLETE indication is raised when all subordinate transaction subtrees, with the possible exception of subordinates from which a TP-HEURISTIC-REPORT indication has been issued, have placed bound data in the final state. This marks the end of the transaction.

If the *commit_return* characteristic has been set to TX_COMMIT_COMPLETED then control is returned to the calling procedure.

C.2 Server Events

Listed below are the server TxRPC CRM events that affect the flow of protocol. In general, only events that affect the flow of protocol are discussed. However, some events which help in establishing context or are useful in furthering the understanding the relationship between the protocol and other components of the TxRPC CRM are also discussed, even though they might not directly cause protocol to flow.

Each event falls into one of three categories:

- events made visible by actions on the TxRPC CRM API
- events made visible by actions on the TX API
- events made visible by the RTI protocol machine.

The events made visible by the TxRPC CRM API for transactional and non-transactional TxRPC are:

- Return (From AP to CRM)
- Unhandled-Exception (From AP to CRM)
- Start-Call (From CRM to AP).

The events made visible by the TX API for transactional and non-transactional TxRPC are:

- *tx_close*( ) (From AP to CRM)
- *tx_open*( ) (From AP to CRM).

In addition, the event made visible by the TX API for transactional TxRPC only is:

- *tx_rollback*( ) (From AP to CRM).

The events made visible by the RTI protocol machine for transactional and non-transactional TxRPC are:

- Internal-Call-Error (issued to the RTI-PM)
- RTI-CALL-TASK indication (raised by the RTI-PM)
- RTI-CALL-RESULTS request (issued to the RTI-PM)
- RTI-CANCEL-CALL indication (raised by the RTI-PM).

In addition, the events made visible by the RTI protocol machine for transactional TxRPC only are:

- RTI-COMMIT-TRANS indication (raised by the RTI-PM)
- RTI-PREPARE-TRANS indication (raised by the RTI-PM)
- RTI-ROLLBACK-TRANS indication (raised by the RTI-PM)
- RTI-ROLLBACK-TRANS request (issued to the RTI-PM)
- RTI-TRANS-COMPLETE indication (raised by the RTI-PM)
- RTI-TRANS-DONE request (issued to the RTI-PM)
- RTI-TRANS-READY request (issued to the RTI-PM).

C.2.1 tx_close()

The *tx_close*( ) event does not cause protocol to flow. It does cause an important state transition.

C.2.2 tx_open()

The *tx_open*( ) event must be issued prior to any other events.

The *tx_open*( ) event does not cause protocol to flow. It does cause an important state transition.

C.2.3 RTI-CALL-TASK indication

An RTI-CALL-TASK indication marks the start of a TxRPC at the server.

An RTI-CALL-TASK indication causes a Start-Call event to be raised. The Start-Call event is defined to be the point when control is transferred to the called procedure (manager function) from the calling procedure. It is important to distinguish between the point when control is transferred from the calling procedure and the point when the called procedure (manager function) receives control since these are necessarily two separate points in time. The Start-Call event is the latter. The Call event is the former.

C.2.4 RTI-CANCEL-CALL indication

An RTI-CANCEL-CALL indication is raised only while a call is in progress at the server. A call is in progress at the server from the point when an RTI-CALL-TASK indication is raised until the point when a Return or Unhandled-Exception is issued.

An RTI-CANCEL-CALL indication causes an Exception to be raised. If the default exception mechanism is overridden (for example comm_status or fault_status is used) the appropriate parameter is populated.

C.2.5 tx_rollback()

The *tx_rollback*( ) event is allowed only while a call is in progress at the server. A call is in progress at the server from the point when an RTI-CALL-TASK indication is raised until the point when a Return or Unhandled-Exception is issued.

The *tx_rollback*( ) event causes the local participant to release all bound data in the initial state and to set the local prepare-time vote to be no. A rollback is not initiated at this time. Instead, when the server (manager function) returns results the TxRPC CRM sets the Transaction Context to indicate that a rollback has been issued in the server and the results are sent to the client. See Section 2.8 on page 15.

C.2.6 RTI-COMMIT-TRANS indication

At least one TxRPC must have been made in order for the RTI-COMMIT-TRANS indication to be raised.

An RTI-COMMIT-TRANS indication is raised when the outcome of a transaction is commitment. After the local participant has placed bound data in the final state an RTI-TRANS-DONE request is issued.

C.2.7 RTI-PREPARE-TRANS indication

At least one TxRPC must have been made in order for the RTI-PREPARE-TRANS indication to be raised.

An RTI-PREPARE-TRANS indication is raised when a transaction is being committed.

If the local participant has issued *tx_rollback*( ) or if the local participant is unable to place all bound data in the ready state, an RTI-ROLLBACK-TRANS request is issued. After the local participant has released bound data an RTI-TRANS-DONE request is issued.

If the local participant has not issued *tx_rollback*( ) and the local participant is able successfully to place all bound data in the ready state, an RTI-TRANS-READY request is issued. The local participant must be in the READY state[22] when this request is issued.

C.2.8 RTI-ROLLBACK-TRANS indication

At least one TxRPC must have been made in order for the RTI-ROLLBACK-TRANS indication to be raised.

An RTI-ROLLBACK-TRANS indication is raised when the current transaction is being rolled back. After all bound data is released in the initial state an RTI-TRANS-DONE request is issued.

22. This state is defined by the OSI TP Model, Service and Protocol standards.

C.2.9 RTI-TRANS-COMPLETE indication

At least one TxRPC must have been made in order for the RTI-TRANS-COMPLETE indication to be raised.

An RTI-TRANS-COMPLETE indication is raised when all subordinate transaction subtrees, with the possible exception of subordinates from which a TP-HEURISTIC-REPORT indication has been issued, have placed bound data in the final state. This marks the end of the transaction.

C.3 Mapping

The tables in this section show a mapping between the TxRPC API or TX API and the protocol. The tables represent a single instance of an entity (client or server) and a transactional TxRPC, in the context of a single transaction.

In general, only events that affect the flow of protocol are discussed. However, some events which help in establishing context or are useful in furthering understanding of the relationship between the protocol and other components of the TxRPC CRM are also discussed.

The tables do not attempt to describe the complete operation of the TxRPC CRM. They are, however, complete and accurate descriptions of the mapping between the APIs and the underlying protocol.

Table C-1 lists the possible output events for the client and Table C-2 on page 162 lists the possible output events and action routines for the server.

In Table C-3 on page 162, Table C-4 on page 163 and Table C-5 on page 164, each row describes a particular event. The first column identifies the event and, where relevant, a second column identifies preconditions that are associated with that event. Every other column describes a state. At the intersection of each event and state are all the actions taken and the state transition which follows these actions. If the event-state pair is not allowed, the intersection is blank. If there are no actions listed but there is a state transition, the event is allowable but there is no action taken, at least not from a protocol and API point of view.

Events that provide input to these tables are described in the previous section.

Table C-1 Client Output Events

Output Events	Definition of Event
Exception	An exception is raised.
Call-Return	Control is returned from an outstanding call.
R-E-C	RTI-ESTABLISH-CONTEXT request
R-C-T	RTI-CALL-TASK request
R-T-C	RTI-COMMIT-TRANS request
R-C-C	RTI-CANCEL-CALL request
R-E-T	RTI-END-TRANS request
R-R-T	RTI-ROLLBACK-TRANS request
R-T-R	RTI-TRANS-READY request
R-T-D	RTI-TRANS-DONE request

Table C-2 Server Output Events and Action Routines

Output Events and Action Routines	**Definition of Event**
Start-Call	A TxRPC is started in the server
Exception	A CMA (POSIX) exception is raised
set vote no	Set the local prepare-time vote to be **no**
I-C-E	Internal-Call-Error
R-C-R	RTI-CALL-RESULT request
R-R-T	RTI-ROLLBACK-TRANS request
R-T-R	RTI-TRANS-READY request
R-T-D	RTI-TRANS-DONE request

Table C-3 API to Protocol Mapping for Non-transactional TxRPCs

Client Events	**Null**	**Open**	**Calling**
Call		R-E-C R-C-T Calling	
Cancel			R-C-C Calling
tx_close( )	Null	Null	
tx_open( )	Open	Open	
RTI_CALL_FAILURE.ind			exception Open
RTI_CALL_RESULTS.ind			return Open
Server Events	**Null**	**Open**	**Called**
Return			R-C-R Open
Unhandled Exception			I-C-E Open
tx_close( )	Null	Null	
tx_open( )	Open	Open	Called
RTI_Call_Task.ind		start-call Called	
RTI_Cancel_Call.ind			exception Called

Table C-4 API to Protocol Mapping for Transactional TxRPCs (Client)

Client Events	Precon-ditions	Null	Open	In-Tx	Calling	Idle	Phase-1	Phase-2	Tx-Done
Call				R-E-C R-C-T Calling		R-C-T Calling			
Cancel					R-C-C Calling				
tx_begin( )			In-Tx						
tx_close( )		Null	Null						
tx_commit( )	Root and local yes vote					R-E-T Phase-1			
	Root and local no vote					R-E-T R-R-T R-T-D Tx-Done			
	Chained			In-Tx					
	¬Chained			Open					
tx_open( )		Open	Open	In-Tx		Idle			
tx_rollback( )						R-R-T R-T-D Tx-Done			
	Chained			In-Tx					
	¬Chained			Open					
RTI-CALL-FAILURE.ind					Exception Idle				
RTI-CALL-RESULTS.ind					Call-Return Idle				
RTI-TRANS-READY.ind	Root and local yes vote						R-T-C Phase-2		
RTI-COMMIT-TRANS.ind	Early Return Chained							R-T-D In-Tx	
	Early Return ¬Chained							R-T-D Open	
	¬Early Return							R-T-D Tx-Done	
RTI-HEURISTIC-REPORT.ind									Tx-Done
RTI-TRANS-COMPLETE.ind	Chained								In-Tx
	¬Chained								Open
RTI-ROLLBACK-TRANS.ind					R-T-D Exception Tx-Done	R-T-D Tx-Done	R-T-D Tx-Done		

Table C-5 API to Protocol Mapping for Transactional TxRPCs (Server)

Server Events	Precon-ditions	Null	Open	Calling	Idle	Phase-1	Tx-Done
Return				R-C-R Idle			
Unhandled Exception				I-C-E Idle			
tx_close( )		Null	Null				
tx_open( )		Open	Open	Calling			
tx_rollback( )				set vote no Calling			
RTI-CALL-TASK.ind			Start-Call Calling		Start-Call Calling		
RTI-CANCEL-CALL.ind				Exception Calling			
RTI-PREPARE-TRANS.ind	local vote yes				R-T-R Phase-1		
	local vote no				R-R-T R-T-D Tx-Done		
RTI-ROLLBACK-TRANS.ind					R-T-D Tx-Done	R-T-D Tx-Done	
RTI-COMMIT-TRANS.ind						R-T-D Tx-Done	
RTI-TRANS-COMPLETE.ind							Open

X/Open CAE Specification

Part 4:
TxRPC ASE Appendices

X/Open Company Ltd.

Appendix D

Mapping to RPC Terminology

This appendix describes the relationship between the service primitive names and conventions used to describe the RPC-ASE as compared with those used in the X/Open **DCE RPC** specification.

D.1 Service Conventions

The X/Open **DCE RPC** specification uses the service primitive class *event* and *action* to describe the relationship to the service user. *Events* are issued by the service user and received by the RPC service provider. *Actions* are issued by the RPC service provider and received by the service user.

The service primitive class *event* in the X/Open **DCE RPC** specification corresponds to the service primitive class *request* in the RPC-ASE description.

The service primitive class *action* in the X/Open **DCE RPC** specification corresponds to the service primitive class *indication* in the RPC-ASE description.

D.2 Service Primitive Names

Table D-1 shows the mapping from the event and action routine names used in the X/Open **DCE RPC** specification to the RPC-ASE primitive name.

Table D-1 Service Primitive Name Mapping

RPC Name	RPC-ASE Primitive Names
TransmitReq event	RPC-REQUEST.req
LastTransmitReq event	RPC-REQUEST.req
LocalAlert event	RPC-REMOTE-ALERT.req
NoConnInd event	RPC-NO-CONN.req
LocalErr event	RPC-ORPHANED.req
HandleFrag action	RPC-REQUEST.ind RPC-RESPONSE.ind
RaiseFault action	RPC-FAULT.ind
SetShutdownRequested action	RPC-SHUTDOWN.ind
RPCResp event	RPC-RESPONSE.req
FaultDNE event	RPC-FAULT.req
Fault event	RPC-FAULT.req
ShutdownReq event	RPC-SHUTDOWN.req
StopOrphan action	RPC-ORPHANED.ind
ProcessAlertMsg action	RPC-REMOTE-ALERT.ind
none	RPC-DONE.ind

Appendix E

Scenarios

This appendix provides examples of RTI usage. The scenarios of RTI usage are not intended to be exhaustive.

E.1 Service Scenarios

The following scenarios are intended to help outline possible protocol exchanges using the RTI Service, and the RTI TP Service.

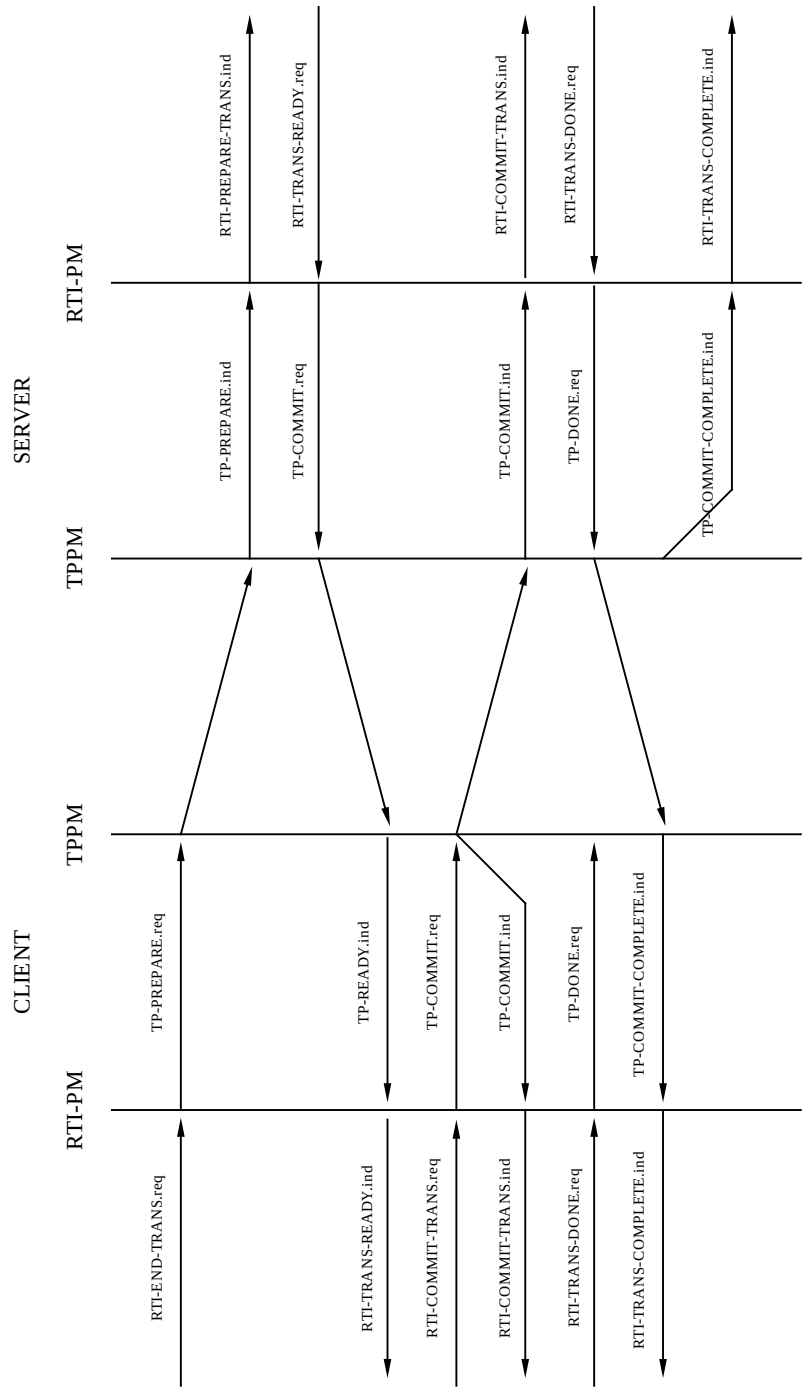


Figure E-1 Client Issues RTI-END-TRANS; Transaction Committed; Active Context Handle

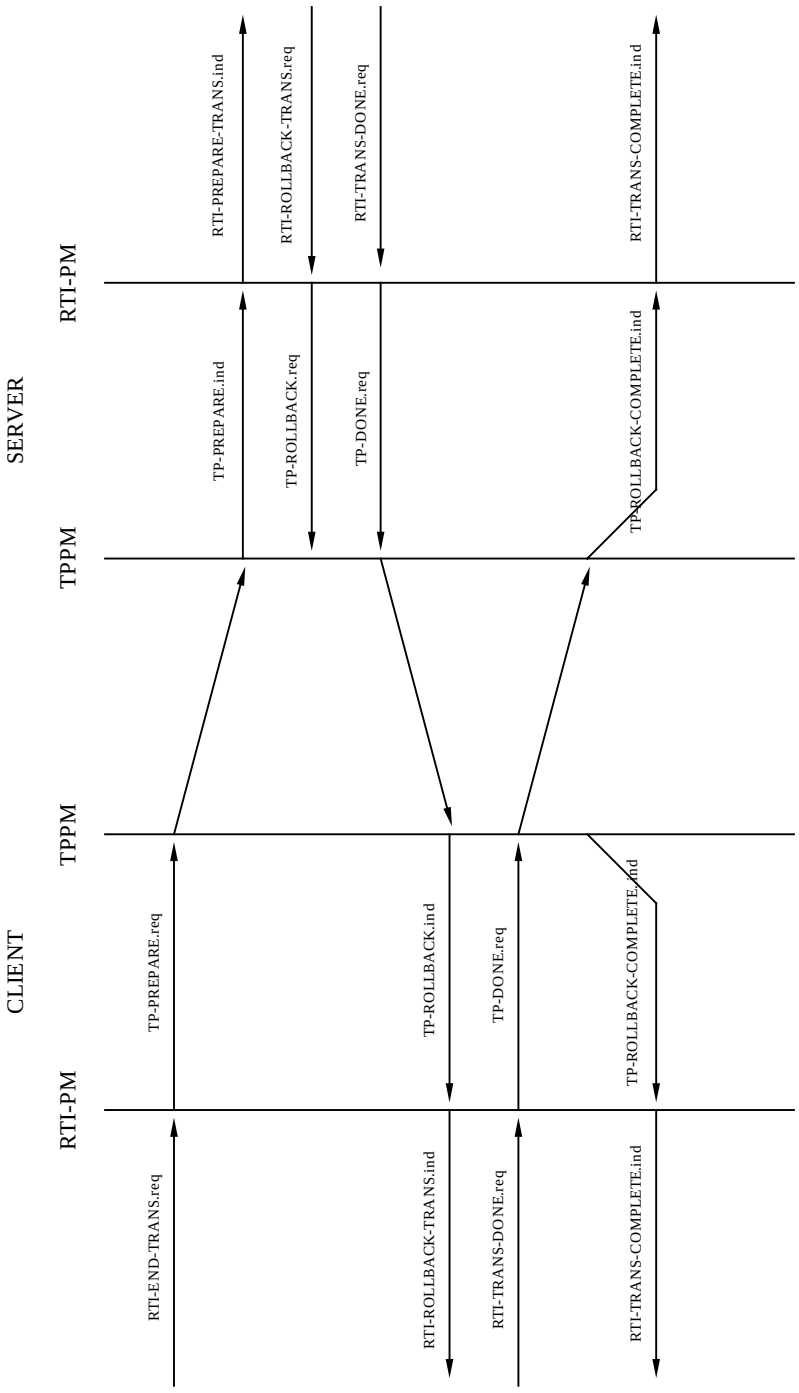


Figure E-2 Client Issues RTI-END-TRANS; Server Issues Rollback; Active Context Handle

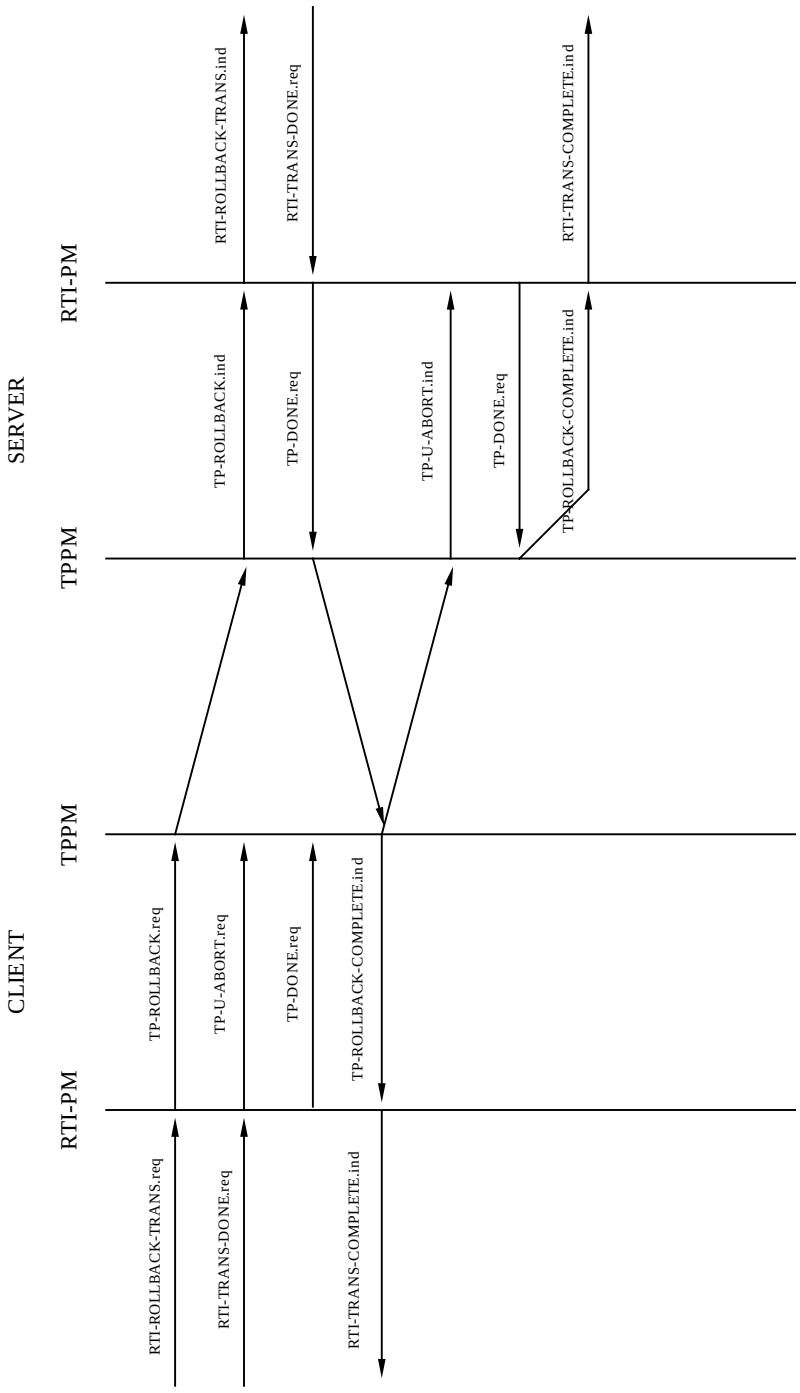


Figure E-3 Client Issues Rollback; No Context Handle

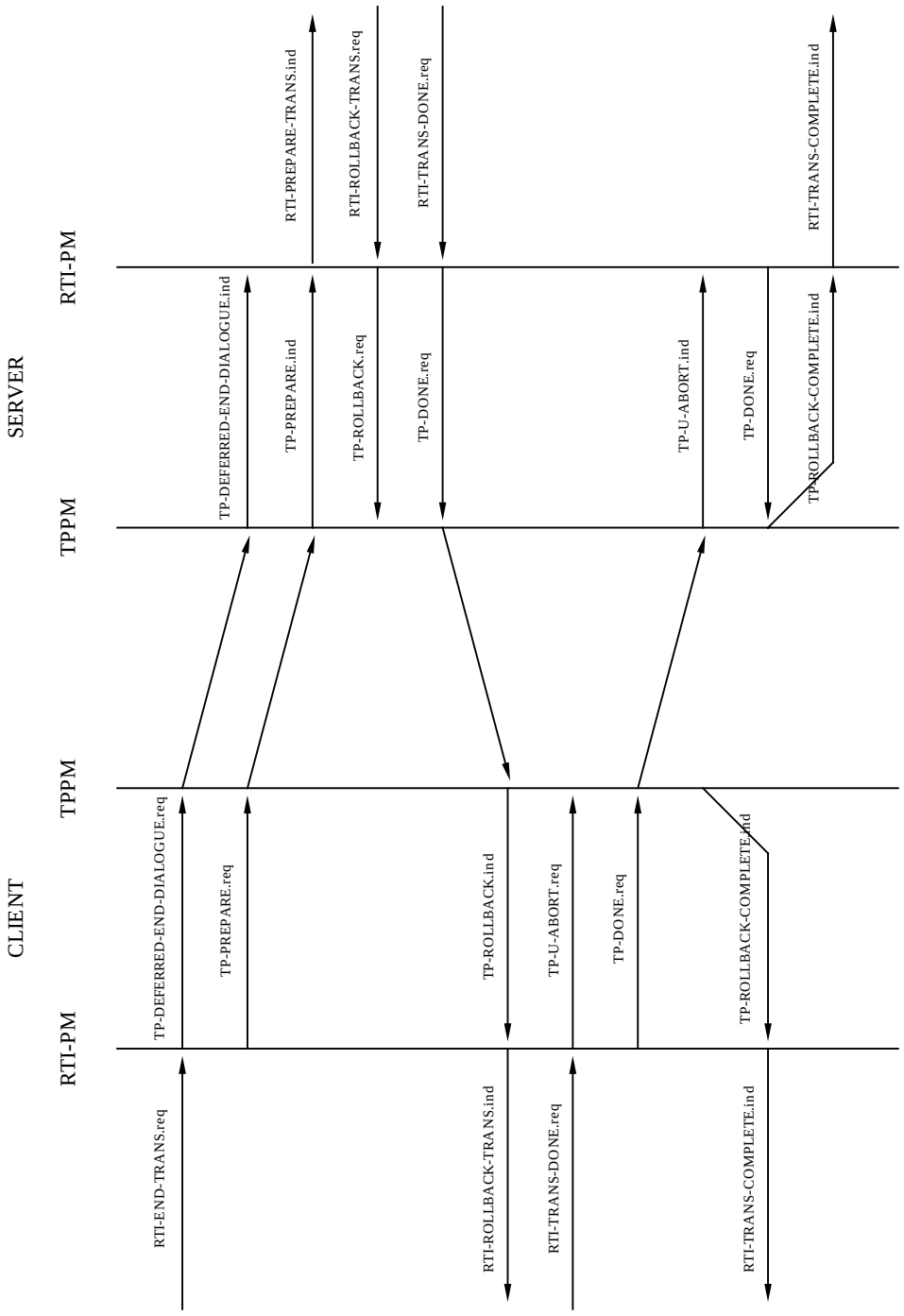


Figure E-4 Server Issues Rollback; No Context Handle

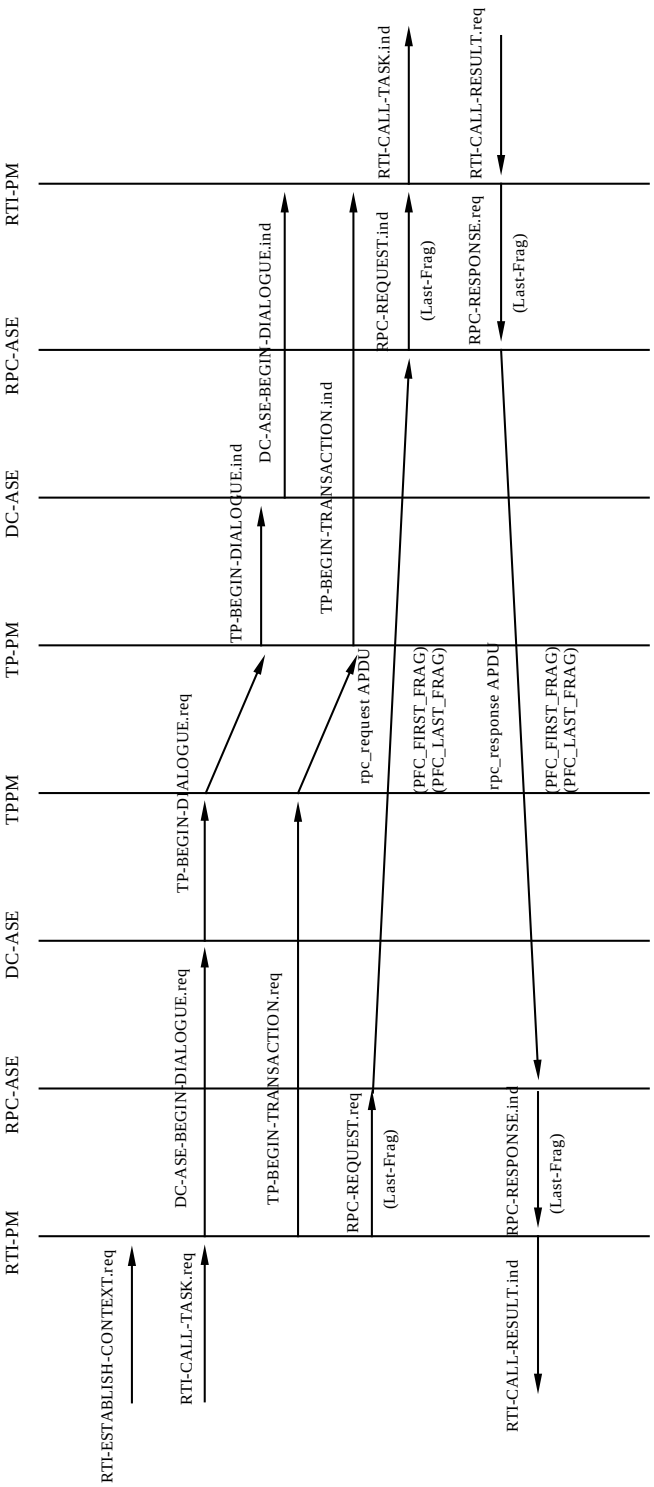


Figure E-5 RPC Request and Response with No Segmentation

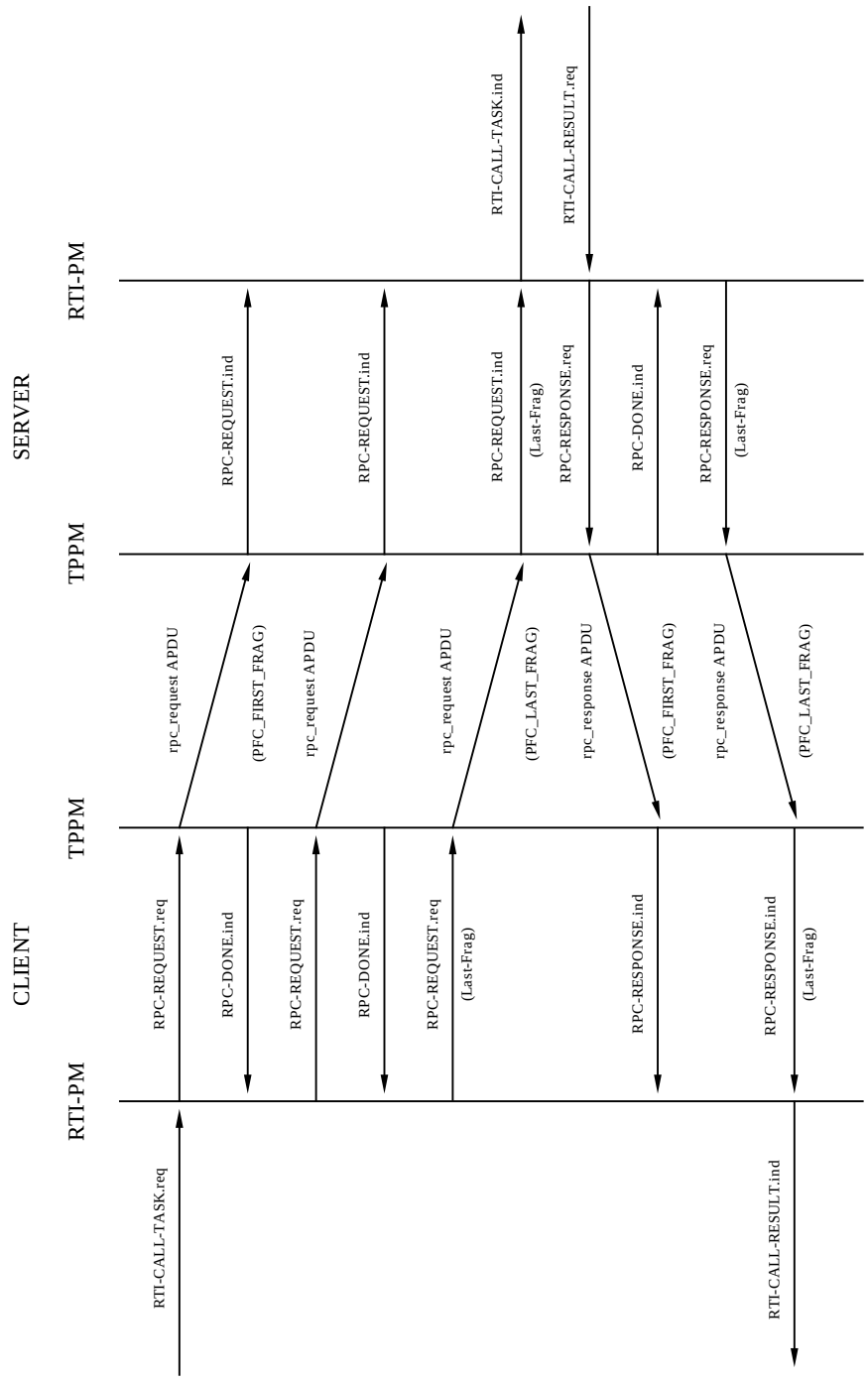


Figure E-6 RPC Request and Response with Segmentation

Glossary

ACSE
Association Control Service Element.

A-Ctx
Application Context.

AE
Application Entity.

AEI
Application Entity Invocation.

AE-Qualifier
Application Entity Qualifier.

ALS
Application Layer Structure.

AP
An Application Process (in the OSI TP Model) or an Application Program (in the X/Open DTP Model).

APDU
Application Protocol Data Unit.

API
An Application Process Invocation (in the OSI TP Model) or an Application Programming Interface (in the X/Open DTP Model).

AP-Title
Application Process Title.

ASE
Application Service Element. (See Part 2 of this specification.)

ASN.1
Abstract Syntax Notation One.

client
A program that issues RPCs. An RTI-SUI requesting a task invocation. The superior of a dialogue. See Section 2.2.7 on page 11.

CRM
Communication Resource Manager.

final state
The state of bound data immediately after the successful completion of a transaction.

IDL
Interface Definition Language. See Section 2.2.10 on page 11.

initial state
The state of bound data at the time it is first accessed by a transaction participant.

ISO
International Organization for Standardization. A standards organisation with the

membership composed of the standards organisations from each participating country. OSI working groups generate the OSI Protocol Suite standards.

MACF
Multiple Association Control Function.

manager function
The AP executed in a server that implements an operation. See Section 2.2.8 on page 11.

Node-ID
The Node-ID is defined to be a globally administered 48-bit address as described in the ISO TP Protocol standard.

Open Systems Interconnection
(OSI) A set of ISO standards that define a network architecture based on a 7-layer model for communication between open systems. OSI management standards define Configuration, Fault, Performance, Security and Accounting Management.

primitive
An event that occurs at an interface between the user of the service and the service provider in an open system.

protocol machine
A finite state machine that generates a particular output, for example, service request, indication, or network activity.

PSDU
Presentation Service Data Unit.

ready state
An intermediate state in which no further modifications are made to bound data before it is declared to be in final state.

recoverable resources
The resource used to store the results of a recoverable operation. Recoverable resources are bound data.

Remote Task Invocation Service Interface
(RTI-SI) The service boundary between the RTI Service User Invocation (RTI-SUI) and the RTI Protocol Machine (RTI-PM) and between the RTI-SU and the RTI-SP.

Remote Task Invocation Service Provider
(RTI-SP) In the RTI Model, a collection of RTI-PMs within an RTI Application Process Invocation (RTI-API).

Remote Task Invocation Service User
(RTI-SU) Any user of the RTI Service. An RTI-SUI is an instance of an RTI-SU.

Remote Task Invocation Service User Invocation
(RTI-SUI) A server that uses the RTI service provided by the RTI Protocol Machine. (RTI-PM) There is one RTI-PM for each RTI-SUI.

RM
Resource Manager.

RPC
Remote Procedure Call. See Section 2.2.6 on page 10.

RTI
Remote Task Invocation.

RTI-AEI
RTI Application Entity Invocation.

RTI-AP
RTI Application Process.

RTI-API
RTI Application Process Invocation.

RTI-MACF
RTI Multiple Association Control Function.

RTI-PM
RTI Protocol Machine.

RTI-SI
Remote Task Invocation Service Interface.

RTI-SP
Remote Task Invocation Service Provider.

RTI-SU
Remote Task Invocation Service User.

RTI-SUI
Remote Task Invocation Service User Invocation.

SACF
Single Association Control Function.

SAO
Single Association Object.

server
A program that accepts RPCs. An RTI-SUI that initiates a task invocation in response to a task invocation request from a client. The subordinate of a dialogue. See Section 2.2.7 on page 11.

TM
Transaction Manager.

TPPM
A TP Protocol Machine in the OSI TP Model.

TPSP
A TP service provider in the OSI TP Model.

TPSU
A TP Service User in the OSI TP Model.

TPSUI
A TP Service User Invocation in the OSI TP Model.

transaction tree
A sub-tree of a dialogue tree in which all the dialogues are transactional dialogues.

transactional RPC
A TxRPC operation initiated from within the scope of a transaction. See Section 2.2.9 on page 11.

TxRPC CRM
An RM that uses the TxRPC interface. See Section 2.2.11 on page 11.

TxRPC operation
An RPC operation that has either the transaction_mandatory or transaction_optional attribute.

universal time constant
(UTC) The value of Universal Coordinated Time as defined in ISO 8601. See the referenced X/Open **DCE RPC** specification.

UUID
Universal Unique Identifier.

version number
The version number of the UUID architecture.

Index