A SPECIALIZATION ON THE REMOTE DATABASE ACCESS

A THESIS IN
Computer Science

Presented to the Faculty of the University
of Missouri-Kansas City in partial fulfillment
of the requirements for the degree

MASTER OF SCIENCE

by
YOUNG BAE CHOI

B.S., Chonnam National University, 1982
M.S., Korea Advanced Institute of Science and Technology, 1985

Kansas City, Missouri
1991

A SPECIALIZATION

ON

THE REMOTE DATABASE ACCESS

Young Bae Choi, Master of Science

University of Missouri-Kansas City, 1991

ABSTRACT

The RDA (Remote Database Access) standards support the interworking between an application program in one open system and a DBMS in a remote open system. The Generic RDA standard defines the common aspects of a class of RDA applications, and a Specialization standard describes a specialization for a particular type of DB in the class.

The RDA model describes RDA via client/server relationship. The services of RDA are grouped into five categories: Dialogue Management, Transaction Management, Control, Resource Handling, Database Language. The server execution rules are defined for each service. The Basic Application Context and TP Application Contexts are used to perform the necessary set of RDA services.

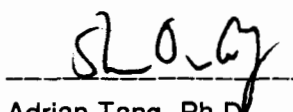
RDA Specialization defines any required constraints on the permissible parameter values for each service. Also, it defines additional entities and their attributes on the Dialogue State Model. For each server operation, additional constraints are defined.

As the basic step of constructing prototype RDA for a subset of database languages, two RDA service user interfaces were designed and necessary functions and parameters were defined. First, to fill the gap of functionalities between the RDA client and the RDA Communications Service, one RDA Client Interface model was designed. Second, an RDA Server Interface model which contains necessary library functions and parameters to send an RDA indication and receive the response for it was designed to fill the gap of functionalities between the RDA server and the RDA Communications Service.

Also, a set of Generic Object Management Library functions for the RDA server as one possible implementation model was defined and the functions for the RDA server to interface with SQL Server Interface for the RDA specialization were refined and added. The internal execution of RDA operation according to the RDA server rules were explained by using the functions of Generic Object Management Library for the RDA server.

All the functions were designed by the object-oriented concept. So, this model can be modified conveniently and implemented easily to accommodate other types of database languages by the object-oriented languages because of functional modularities of library functions.

This abstract of 341 words is approved as to form and content.



Adrian Tang, Ph.D.

Computer Science Telecommunications Program

The undersigned, appointed by the Director of Computer Science Telecommunications Program, have examined a thesis entitled " A Specialization on the Remote Database Access" presented by Young Bae Choi, candidate for the Master of Science, and hereby certify that in their opinion it is worthy of acceptance.

SLOWJ — 12/16/91

Adrian Tang, Ph.D. — Date

Computer Science Telecommunications Program

Jonathan OH — 12/19/91

Jonathan C. Oh, Ph.D. — Date

Computer Science Telecommunications Program

Shen, Xiaojun — 12/16/91

Xiaojun Shen, Ph.D. — Date

Computer Science Telecommunications Program

TABLE OF CONTENTS

	Page
ABSTRACT	ii
LIST OF FIGURES	vii
LIST OF TABLES	ix
LIST OF ABBREVIATIONS	x
GLOSSARY	xi
ACKNOWLEDGEMENTS	xiii
I. INTRODUCTION	1
II. GNERIC RDA SERVICES	6
2.1 Dialogue Management Services	7
2.2 Transaction Services	8
2.3 Control Services	10
2.4 Resource Handling Services	10
2.5 Database Language (DBL) Services	12
III. PROTOCOL	14
3.1 Dialogue State Model	14
3.2 General Server Execution Rules	16
3.3 Server Rules for Each Service Element	18
IV. RDA APPLICATION CONTEXTS	25
4.1 Basic Application Context	25
4.1.1 SACF Rules	26
4.2 TP Application Context	29
4.2.1 SACF Rules	30
V. SPECIALIZATION	38
5.1 Introduction	39
5.2 RDA SQL Specialization Service Model	40
5.2.1 Mapping to the General Model of the RDA Service	40
5.2.2 Mapping to the Concepts of Database Language SQL	42
5.3 RDA SQL Specialization Services	42
5.3.1 Dialogue Management Services	42

5.3.2 Transaction Management Services 43
5.3.3 Control Services 44
5.3.4 Resource Handling Services 44
5.3.5 Database Language Services 45
5.3.6 Sequencing Rules 48
5.4 RDA SQL Specialization Data Types 48
5.4.1 RDA SQL DBL Operations 48
5.4.2 RDA SQL DBL Arguments and Results 49
5.4.3 RDA SQL DBL Descriptors 51
5.4.4 RDA SQL DBL Exceptions 54
5.5 Protocol 55
5.5.1 Dialogue Management Services 55
5.5.2 Transaction Management Services 56
5.5.3 Control Services 56
5.5.4 Resource Handling Services 56
5.5.5 Database Language Services 57
5.5.6 Structure and Encoding of RDA SQL APDUs 63

VI. RDA SERVICE INTERFACE AND SERVER MODULES DESIGN .. 65

6.1 Overview 65
6.2 RDA Client Interface 66
6.2.1 Design Model 67
6.2.2 Library Functions and Parameters 68
6.3 RDA Server Interface 75
6.3.1 Design Model 75
6.3.2 Library Functions and Parameters 76
6.4 RDA Server Modules 83
6.4.1 Design Model 83
6.4.2 Interface to the SQL Server 88
6.4.3 Generic Object Management Library Functions and Parameters 88
6.4.4 Server Operation Execution for Each Service 96

VII. CONCLUSIONS 112

APPENDIX 114
REFERENCES 139
VITA 140

List of Figures

Figure		Page
Figure 1-1	The Basic RDA Model	2
Figure 1-2	The RDA Dialogue States	4
Figure 1-3	Query Processing in the RDA Environment	5
Figure 4-1	Two RDA Application Contexts	25
Figure 4-2	State Transition Diagram for RDA Basic Application Context - Server	28
Figure 4-3	State Transition Diagram for RDA Basic Application Context - Server	29
Figure 4-4	State Transition Diagram for TP Application Context - Client (Chained Transactions)	34
Figure 4-5	State Transition Diagram for TP Application Context - Server (Chained Transactions)	35
Figure 4-6	State Transition Diagram for TP Application Context - Client (Unchained Transactions)	36
Figure 4-7	State Transition Diagram for TP Application Context - Server (Unchained Transactions)	37
Figure 5-1	Referencing of the Embedded SQL Variables	51
Figure 6-1	RDA Model	66
Figure 6-2	The Structure of RDA Client Interface	68
Figure 6-3	The Structure of RDA Server Interface	76
Figure 6-4	State Transition of Operation in the RDA Server	84
Figure 6-5	R-Initialize Service Operation Execution	97
Figure 6-6	R-Terminate Service Operation Execution	98
Figure 6-7	R-BeginTransaction Service Operation Execution	99
Figure 6-8	R-Commit Service Operation Execution	100
Figure 6-9	R-Rollback Service Operation Execution	101
Figure 6-10	R-Cancel Service Operation Execution	102
Figure 6-11	R-Status Service Operation Execution	103

Figure 6-12 R-Open Service Operation Execution 104

Figure 6-13 R-Close Service Operation Execution 105

Figure 6-14 R-ExecuteDBL Service Operation Execution 106

Figure 6-15 R-DefineDBL Service Operation Execution 107

Figure 6-16 R-InvokeDBL Service Operation Execution 108

Figure 6-17 R-DropDBL Service Operation Execution 109

List of Tables

Table			Page
Table	2-1	Service Element in Dialogue Initialization Functional Unit	7
Table	2-2	Service Element in Dialogue Termination Functional Unit	8
Table	2-3	Service Elements in Transaction Management Functional Unit	9
Table	2-4	Service Elements in Control Services	10
Table	2-5	Service Elements in Resource Handling Services	11
Table	2-6	Service Elements in Database Language Services	12
Table	3-1	Server Rules for the Dialogue Management Services	19
Table	3-2	Server Rules for the Transaction Management Services	20
Table	3-3	Server Rules for the Control Services	21
Table	3-4	Server Rules for the Resource Handling Services	22
Table	3-5	Server Rules for the DBL Services	24
Table	5-1	RDA Services for the SQL Specialization	42
Table	5-2	Use of SQL Argument and Result Parameters in R-ExecuteDBL Service	59
Table	5-3	Use of SQL Argument and Result Parameters in R-DefineDBL Service	61
Table	5-4	Use of SQL Argument and Result Parameters in R-InvokeDBL Service	63

List of Abbreviations

AC	Application Context
ACID	Atomicity, Consistency, Isolation and Durability
ACSE	Association Control Service Element
AE	Application Entity
AEI	Application Entity Invocation
ANSI	American National Standards Institute
AP	Application Process
APDU	Application PDU
ASE	Application Service Element
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules (ASN.1/OSI/CCITT)
CCR	Commitment Concurrency & Recovery protocol
DB	Database
DBL	Database Language
DBMS	Data Base Management System
DDL	Data Definition Language
DIS	Draft International Standard (proposed ISO standard)
DML	Data Manipulation Language
DP	Draft Proposal
IA5	International Alphabet number 5 (V.3/CCITT)
IS	International Standard (final stage of OSI protocols definition)
ISO	International Organization for Standardization
IT	Information Technology
JTC	Joint Technical Committee
MACF	Multiple Association Controlling Function
OSI	Open System Interconnection
RDA	Remote Database Access
RO-Notation	Remote Operation Notation
SACF	Single Association Controlling Function
SAO	Single Association Object
SC	Subcommittee
SQL	Structured Query Language
TC	Technical Committee (ISO/ECMA)
WD	Working Document
WG	Working Group

Control Operation: An RDA Operation that does not affect the DB directly but which references other RDA Operations.

Database Language: A language which is used to define the syntax and semantics of operations on a database.

Database Language (DBL) Command: An RDA operation type which models a request for the database access and update. A Database Language (DBL) Command has a command handle, a DBL statement, and optional argument and optional result formal parameters.

Database Language Statement: The definition of an operation on a database in a Database Language.

Database Management System: A software system within the server that manages the persistence and integrity of the database and provides database language services to its users.

Database Server: An open system which provides the facilities for database storage and supplies database processing services to other open systems.

Data Resource: A named collection of data and/or capabilities on the Database Server known to both the RDA Server and the RDA Client. The RDA Client must open a Data Resource to get access to its data content or capabilities. Further properties of Data Resource may be defined by RDA Specializations.

Generic Application Context: An incomplete definition of an Application Context, which defines the common properties of a class of application contexts.

Generic Application Service Element: An incomplete definition of an ASE, which defines the common properties of a class of ASEs.

Generic RDA Service: A template for the Specific RDA Services definition. It describes the common properties of a class of specific RDA services.

RDA Abstract Service Interface: The conceptual boundary between the RDA Communication capabilities and the non-communication capabilities within the RDA client and RDA Server Open Systems.

RDA Client: The RDA Communications Service User that initiates an RDA dialogue and requests database access from a remote Database Server.

RDA Client Interface: The boundary between the RDA Communications Service and the RDA Client. This is a part of the RDA Service Interface.

RDA Communications Service: A service which provides the interworking between an RDA Server and an RDA Client.

RDA Control Dialogue: A dialogue which is used for communication of Control Operations for another dialogue.

RDA Control Service: An RDA operation type which controls the outstanding RDA operations. This service allow an RDA Client to query the RDA Server for the status of particular outstanding operations and to cancel the outstanding operations.

RDA Dialogue: A cooperative relationship between n RDA Server and an RDA Client. It has a unique identifier that is assigned by the RDA AEI when the RDA Dialogue is established.

RDA Dialogue State Model: A data structure which models the state of an RDA Dialogue. This model is defined by a set of entity types and their attributes.

RDA Operation: A request for processing which is initiated by the RDA Client. This is used for specifying the behavior of the RDA Server.

RDA Protocol Machine: The protocol machine of an RDA Application Service Element.

RDA Server: The RDA Communications Service User in a Database Server which provides database access to remote RDA Clients.

RDA Server Interface: The boundary between the RDA Communications Service and the RDA Server.

RDA Service Interface: The conceptual boundary between the RDA communications capabilities and the non-communication capabilities in the RDA Client and RDA Server open systems.

RDA Specialization: A standard which elaborate a usage of the Generic RDA Service for a particular type of database. This is characterized by a Database Language as SQL.

RDA Transaction: A logically complete unit of processing as determined by the RDA Client. This is used to guarantee the consistency of remote database processing.

Specific RDA Service: A service defined by specializing a Generic RDA Service. This defines a member of the class of services and has all the properties of the Generic RDA Service plus the extra properties making it appropriate for use in a specific application.

SQL Data Resource: An SQL DB as defined in ISO/IEC 9075 (SQL).

SQL Server: A conforming implementation of ISO/IEC 9075 (SQL) which provides the DB services at the RDA Server.

Acknowledgements

I am happy to take this opportunity to thank all the people who helped me in big and small ways in the preparation of the thesis.

I thank Dr. Adrian Tang for introducing me to the thesis topic and many helpful and continuous comments. I also thank Dr. Jonathan C. Oh and Dr. Xiaojun Shen for their working as the committee members and giving me the useful comments in many ways.

I thank Dr. Joel S. Berson, the editor of the standard documents on the Remote Database Access (RDA), for his contribution on the resolution of my questions.

Finally, I wish to represent my deep-hearted thanks to my wife [redacted] and daughter [redacted] for their precious sharing of their lives during my study.

Note: At the request of the author, the names of family members have been redacted from this digitized copy.

CHAPTER I

Introduction

Nowadays, the **Distributed databases (DDBs)** play an important role in a computer integrated manufacturing environment. In a manufacturing environment, it is typical to find a set of databases existing at several levels. Accordingly, the need to integrate these systems is growing. **Distributed database management systems (DDBMSs)** provide the high level functionality needed while **Remote Database Access (RDA)** provides the communications mechanism for integrating the different database (DB) systems. The services of RDA allow a user to use the same front-end system to access various DBs, so that a single DB can be shared by the different users with different workstations and various man-machine interfaces. If the user cannot use the RDA, the user must handle different user interfaces to different DB systems. Also, the supply of robust workstation software which has different capabilities for each workstation by the DB vendors becomes not easy. With the help of RDA, an application running in a workstation in one open system can interwork with a DB system in a remote open system .

The development of RDA standard lags behind that of standards for many other application protocols. The RDA standard, as documented in ISO/IEC 9579, is still at a Common Proposal (CP) stage. This document is made up of two parts. Part 1, the **Generic RDA standard** (ISO/IEC 9579-1), defines the common aspects of a class of RDA applications. **Specialization standard** for each different type of DB access in the class is under development. A Specialization standard defines how the generic standard can be specialized for a particular DB access. So, the subclasses are defined in a Specialization standard. At present, the only specialization standard is for SQL which is found in part 2 of ISO/IEC 9579 (ISO/IEC 9579-2). Thus likely ISO/IEC 9579 will contain more than two parts in the future.

In this chapter, we introduce some RDA modelling concepts such as the Dialogue State Model.

The **basic RDA model** is a client-server relationship. Figure 1 shows the basic RDA model and its component relationships. A DB is controlled by the **Database Server** which is an open system that provides DB storage facilities and supplies DB services to other open systems. An **RDA Communications Service** is responsible for the interworking between an RDA client and an RDA server. The **RDA client**, modelled by an application process running in some workstation, needs to read or update the data of the DB to complete its processing job(s). The

client initiates an RDA dialogue and requests RDA operations to be executed by the RDA server while the **RDA server**, being passive, responds to the client requests by providing DB services to RDA clients. The RDA server is in the Database Server. Both the client and the server are the RDA service users. The boundary between the RDA server and the RDA Communications Service is called the **RDA Server Interface**. The boundary between the RDA client and the RDA Communications Service is called the **RDA Client Interface**. The client interface can be asynchronous or synchronous. In the asynchronous interface, the client can invoke more operations without waiting for a result from a previous invocation. Sometimes, the application will require the client to wait for a confirm, although this is not the concern of the RDA service.

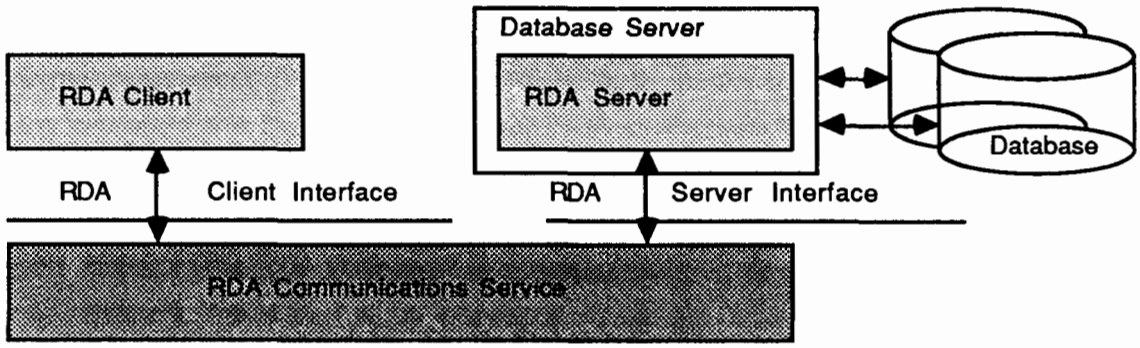


Figure 1-1 The Basic RDA Model

First, there are several concepts which are necessary to understand the Database Server. In the organization of data, a named collection of data and/or capabilities on the Database Server, that is a **Data Resource** is used as a basic data unit known to both the RDA client and the RDA server. The client must open a Data Resource to use it in subsequent request for Database Language Services or as a parent data resource in subsequent R-Open requests. Subordinate Data Resource may be grouped in their parent resources. A parent resource must be opened to open the subordinate resources. The semantics of data content and capabilities of a Data Resource depend upon the applications of RDA which is determined by each RDA Specialization standard.

In the processing by Database Server, the cooperative relationship between an RDA server and RDA client is modelled by an **RDA dialogue** which a unique identifier is assigned when the dialogue is first established -- note that we also use the term dialogue to model a relationship between two TPSUIs in TP. A dialogue can last for more than one application association. All interactions between the RDA server and the RDA client occur in the context of an RDA dialogue, so it can be considered as a liason between the server and the client. If any of the RDA server, RDA client, or application association fails when the dialogue is active, the the

RDA dialogue will fail, too. A dialogue failure cancels the results of any transaction(s) which have not yet been successfully committed. When a dialogue is active, the client may invoke the operations while waiting for the results of previous RDA operations. So, it is possible several outstanding operations exist for a single RDA dialogue. The **RDA Dialogue State Model** is a data structure which models the state of an RDA dialogue. This model is defined by a set of entity types and their attributes. **RDA transaction** is a logically complete unit of processing as decided by the client. At any time at most one RDA transaction is being processed in the one RDA dialogue, though there may be several RDA transactions being processed concurrently in the DB server for different RDA dialogues. **Database Language (DBL) Commands** is an interaction type between the RDA server and the RDA client for accessing or updating the DB. A DBL command is composed of a command handle, a DBL statement, and optional argument and optional result formal parameters. An RDA client can define or invoke a DBL command in a single RDA operation, or supply the DBL command to the RDA server and then invoke it later using another RDA operation. An **RDA operation** models a request for processing by the RDA client. The **execution** effects of RDA operations are specified as insertion or deletion of entries or updates to attribute values in the RDA Dialogue State Model.

Second, there are some concepts which are necessary to understand the RDA communications.

The RDA service and protocol are part of the Application Layer of the **OSI** Basic Reference Model. An **RDA service** is a structure of service primitives which are events at the RDA Client Interface or the RDA Server Interface. It models an interaction between an RDA server and an RDA client. The RDA operations status may be determined or their execution may be cancelled by the **Control services**. The RDA provides one-phase commit **transaction management** by the RDA service's Transaction Management functional unit and two-phase commit transaction management by the TP Service's Commit functional unit. **Failure** is the inability of an RDA dialogue to function properly. **Recovery** is the process of restoring a failed RDA dialogue to a consistent and valid state to continue the operations correctly. **Suspension** is the planned, temporary halting of an RDA dialogue. **Resumption** is the continuation of an RDA dialogue after suspension. An RDA dialogue can exist only in the context of an established Application Association, and will not exist if the association is released. An RDA dialogue can be in one of 2 dialogue states:

* **Inactive**: There is no RDA dialogue between the server and the client.
* **Active**: There is an RDA dialogue established between the server and the client.

The Active state can be refined further into 3 transaction states:

* **Transaction Not Open:** No RDA transaction is in progress.
* **Transaction Open:** An RDA transaction is in progress, but it is not in the process of terminating.
* **Transaction Terminating:** An RDA transaction is progress, but it is in the process of being terminated to commit or roll back the state of the DB.

Figure 1-2 shows the RDA dialogue states.

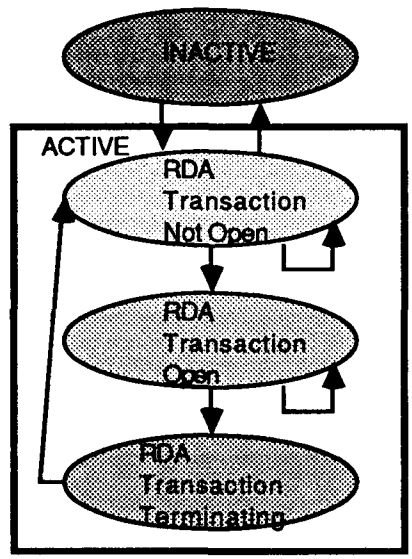


Figure 1-2 The RDA Dialogue States

The **RDA Protocol Data Units (PDUs)** are defined by Abstract Syntax Notation One (ASN.1). These definitions must be refined by the definitions of a specific RDA service.

Third, the **query processing** in the RDA environment, as viewed by a client, can be described as follows (Figure1-3). The calling client submits a query with a specific database language (DBL) from its workstation. The DBL performs lexical and syntactic analysis. The remote DB administrator (DBA) supplies the facility for the communications across the RDA server and the client. The server performs semantic analysis and query optimization. In the returning path, the server performs data computations, table joins, and table projections. The workstation then shows the result on the screen. The service interface on the client side may be provided by a component of a local DBMS so that the user interface for remote access to data need not differ substantially from that used for access to local data.

In the following Chapter II, we cover the Generic RDA Services. In Chapter III, we describe the rules that an RDA server uses when serving the RDA service elements. In Chapter IV, we examine the two application contexts used in conjunction with the RDA ASE, i.e., the Basic Application Context and TP Application Context. In Chapter V, we cover the RDA Specialization on SQL. In Chapter VI, a design of RDA Service Interface will be discussed. Finally, the conclusion will be given in Chapter VII.

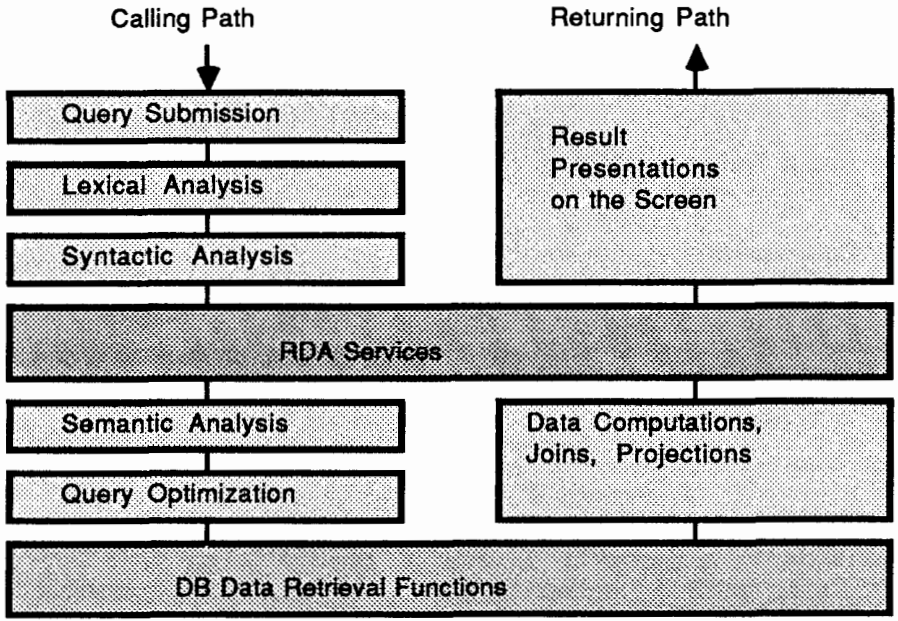


Figure 1-3 Query Processing in the RDA Environment

CHAPTER II

Generic RDA Services

In this chapter, the definitions of the Generic RDA services will be described. It is a template for the definition of Specific RDA services. The RDA services are existing to enable the RDA server and the RDA client to exchange information in an orderly manner.

The RDA services will be explained hierarchically according to the following order:

* Classification of the services into groups according to their function in supporting the client's requests for:

  - RDA dialogue management;
  - RDA transactions management;
  - Control of outstanding operations;
  - Control of data resources availability; and
  - Definition and invocation of server DB operations.

* Classification of the services in each group into functional units according to their ability to be used independently of other functional units.

* Explanation of the service element(s) in each functional unit, including the major parameters for each service element.

Dialogue Management Services provide the facilities for the management of an RDA dialogue with an RDA server. **Transaction Management Services** support the management of RDA transactions which are used to change the DB from one consistent state to another consistent state. **Control Services** are used for the cancellation and determination of outstanding RDA operations. **Resource Handling Services** manages the Data Resources. **Database Language (DBL) Services** are concerned with the definition and dropping of DBL operations, the invocation of already defined operations and the execution of DBL operations.

Because all the RDA service elements can be modelled by abstract operations, we can divide the primitives of each service element into a request group, a result group and an error group.

2.1 Dialogue Management Services

The Dialogue Management Services contains two functional units, Dialogue Initialization Functional Unit and Dialogue Termination Functional Unit. The **Dialogue Initialization Functional Unit** allows a client to start a dialogue with a server. There is one service element in the Dialogue Initialization Functional Unit: R-Initialize (Table 2-1).

Functional Unit Name	Service Elements
Dialogue Intialization Functional Unit	R-Initialize

Table 2-1 Service Element in Dialogue Initialization Functional Unit

The **R-Initialize** service element establishes a new RDA dialogue between the RDA server and the RDA client. During the creation of a new dialogue the requesting RDA client can negotiate several parameters. The request parameter dialogueID is used to uniquely identify the dialogue being created in the OSIE. The operationID parameter is to uniquely identify the operation within the dialogue among all outstanding operations. The identityOfUser parameter shows a character string representing a name or other identifier known to the client and the server. It is used to identify the user, and may be used by the server to determine privileges for the dialogue. The parameter controlServiceDataRequested is a Boolean parameter which indicates whether or not the client desires the return of authentication data required to use Control services on another dialogue. The parameter functionalUnitsRequested is used by the client to request its desired functional units:

* TERMINATION - Dialogue Termination
* TRANSACTION - Transaction Management
* CANCEL - Cancel
* STATUS - Status

* RESOURCE - Resource Handling
* IMMEDIATE-DBL - Immediate Execution DBL
* STORED-DBL - Stored Execution DBL

The result parameter specificInitializeResult and the error parameter specificInitializeError are defined by each RDA Specialization. If the operation is aborted, the error parameter OperationAborted will show the information by two subparameters of it, errorType and diagnosticInformation.

The **Dialogue Termination Functional Unit** allows a client to terminate the existing dialogue. There is one service element in the Dialogue Termination Functional Unit: R-Terminate (Table 2-2). The **R-Terminate** service element orderly terminates a dialogue between the RDA client and the RDA server. All the opened resources are closed at the end of this service element and all defined DBL statements are deleted. A dialogue cannot be terminated within a transaction. The request parameter, specificTerminateArgument and the result parameter specificTerminationResult are defined by each RDA specialization. If an RDA Specialization-defined error was occurred during the processing of an R-Terminate service request, the error parameter SpecificTerminateError will show the relevant information. This may also include supplementary information whose meaning and format is defined by each RDA Specialization.

Functional Unit Name	Service Elements
Dialogue Termination Functional Unit	R-Terminate

Table 2-2 Service Element in Dialogue Termination Functional Unit

2.2 Transaction Management Services

Transaction management depends on the application context -- RDA application contexts will be described in the Chapter IV. RDA provides two application contests -- the RDA Basic Application Context and RDA TP Application Context. For the RDA Basic Application Context, it is provided by the RDA service, while for the RDA TP Application Context, it is provided by the TP service. At any time, at most one RDA transaction is being processed within

one dialogue. But several transactions may be in process concurrently within the DB server for different dialogues. The changes to data resources during a transaction are not known to the other clients until the transaction completes. In case of a failure occurring during the transaction, it will be aborted and any changes made will not be applied to the DB.

In between the states the DB might seen as inconsistent by the another application, however, the inconsistency is only an application concept and will not concern the DB system. The application can inform the DB server when a transaction is being started and terminated.

There are 2 approaches to bulk transfer in OSI. The first one is the **File Transfer approach** and it works like a pipeline. The mechanisms for regulating data flow are the check points and restart. The second approach is the **Reliable Transfer Service approach.** This uses store and forward technique so that the user does not have to establish a virtual circuit between the source and destination before starting the transfer. This is better than the first one.

There are 3 service elements in the **Transaction Management Functional Unit**: R-BeginTransaction, R-Commit and R-Rollback (Table 2-3). This functional unit is only supported by the RDA Basic Application Context.

Functional Unit Name	**Service Elements**
Transaction Management Functional Unit	R-BeginTransaction, R-Commit, R-Rollback

Table 2-3 Service Elements in Transaction Management Functional Unit

The **R-BeginTransaction** service element allows a client to begin a new transaction. It may be used only if there is no transaction currently in progress.

The **R-Commit** service element is used by the client to indicate that the current transaction should be completed by committing the transaction if possible. No further operations may be generated by a client after issuing this service request until a confirmation of commit or rollback has been received. One of the result parameters, transactionResult indicates the disposition of the transaction which is either ROLLEDBACK or COMMITTED.

The **R-Rollback** service element is used by the client to indicate that the current transaction should be completed by performing a rollback. Once a client issues this service request, no further operations may be issued until the confirmation has been received.

2.3 Control Services

The execution of RDA operation by the Server is not instantaneous and, because of the asynchronous characteristic of RDA, several operations may be outstanding at one time. So, the control services allowing a client to query the DB server for the status of a particular outstanding operation or to cancel outstanding operations are necessary. Two functional units of control services are defined: Cancel Functional Unit and Status Functional Unit (Table 2-4).

Functional Unit name	Service Elements
Cancel Functional Unit	R-Cancel
Status Functional Unit	R-Status

Table 2-4 Service Elements in Control Services

The **Cancel Functional Unit** contains only one service element, the R-Cancel service element. The **R-Cancel** service element is used to cancel all or a selected set of outstanding operations. It is a confirmed service. The cancel service response may be returned before the results of previously issued service requests. The request parameter operationsReferenced is a required parameter that shows which operations are to be cancelled. Only Resource Handling Service and Database Language Service operations are cancelled. It consists of one of 2 values: allOperations or listOfOperationsID. The specificCancelArgument is defined by an RDA specialization. The result parameter specificCancelResult is defined by each RDA specialization.

The **Status Functional Unit** contains only one service element, the R-Status service element and is used to determine the status of outstanding RDA operations. The **R-Status** service element determines the status of one or more outstanding operations. The R-Status service response may be returned before the previously issued service request results. The operationStatus, one of the result parameters, may be in the following status: operationIDUnknown, awaitingExecution , executing, finished, cancelled and aborted.

2.4 Resource Handling Services

This service is used to manage Data Resources. The data available at an RDA Server is organized into a set of data resources which access to the content of a data resource requires that it be opened first. The resources which are subject to R-Open and R-Close are defined by implementors, subject to constraints that will be defined in each RDA Specialization standard. The **Resource Handling Functional Unit** is used to manage Data Resources. It has 2 service elements: R-Open and R-Close (Table 2-5).

Functional Unit name	Service Elements
Resource Handling Functional Unit	R-open, R-Close

Table 2-5 Service Elements in Resource Handling Services

The **R-Open** service element is used to identify a data resource which will be accessed in succeeding service requests. A successful R-Open operation makes a data resource available and may limit its availability to the other DB users. The client provides a data resource handle or identifier to reference the opened data resource in Database Language service requests and as a parentDataResourceHandle in further R-Open operations. One of the request parameters, dataResourceHandle, is a name used by Database Language and Resource Handling Services to identify the data resource. Within the scope of the RDA dialogue, this handle value must be unique with respect to the other currently valid data resource handles. The parentDataResourceHandle is the handle of the data resource referenced as the parent data resource if exist. The dataResourceName is used to identify a resource in the underlying DB for opening purposes. The specificUsageMode parameter is a requested access mode(s) to the specified data resource which is defined by each RDA specialization.

The **R-Close** service element is used to terminate the availability of one or more data resources, and of DBL command handles referencing these resources. The result of issuing an R-Close service in an RDA Transaction is defined by each specialization. The meaning and structure of request parameter, specificCloseArgument, is defined by each RDA specialization. The result parameter, list of closeExceptions, contains the parameter closeException which identifies the nature of the close exception. The parameter closeException can have the value either DataResourceHandleUnknown or specificCloseException.

2.5 Database Language (DBL) Services

Database Language Services perform Database Language operations. The Database Language is defined in, or referenced from, each RDA Specialization. The RDA Generic standard does not contain the function of DBL operations at the Database Server. A DBL Statement defines an operation on the DB which is performed at the request of the user. A DBL Statement may be provided and executed by an R-ExecuteDBL operation or may be provided by an R-DefineDBL operation and executed by an R-InvokeDBL operation. DBL services contains 2 functional units: Immediate Execution DBL Functional Unit and Stored Execution DBL Functional Unit (Table 2-6).

Functional Unit Name	Service Elements
Immediate Execution DBL Functional Unit	R-ExecuteDBL
Stored Execution DBL Functional Unit	R-DefineDBL, R-InvokeDBL, R-DropDBL

Table 2-6 Service Elements in Database Language Services

The **Immediate Execution DBL Functional Unit** contains the **R-ExecuteDBL** service element which is used to execute a single DBL statement a fixed number of times and to inform the client of its result. The operation may update the state of the DB. The request parameter specificDBLStatement contains a DBL statement which defines the Database Server operation to be executed together with the specificDBLArgumentSpecification and specificDBLResultSpecification. The details of this parameter and the rules for determining the corresponding specificDBLArgumentSpecification and specificDBLResultSpecification will be defined by each RDA specialization. The parameter specificDBLArgumentSpecification defines the data types of the the argument of a DBL statement which is defined in detail by each RDA specialization. The parameter specificDBLResultSpecification defines the data types of the result of a DBL statement which is defined in detail by each RDA specialization. The dBLArguments, containing the singleArgument or multipleArgument, shows whether or not the

specificDBLStatement is executed more than once. If a singleArgument is specified, then the specificDBLStatement will be executed repetitionCount times with specificDBLArgumentValues. If the multipleArgument containing n specificDBLArgumentValues is specified, then the specificDBLStatement will be executed n times with each specificDBLArgumentValues once. If neither one is specified, then the specificDBLStatement will be executed exactly once. The specificDBLArgumentValues carries the actual parameters for a sequence of single DBL statement requests.

The **Stored Execution DBL Functional Unit** is used by a client to define and store and execute DBL operations later on. There are three service elements in Stored Execution DBL Functional Unit. They are R-DefineDBL, R-InvokeDBL, and R-DropDBL. These service elements allow a client to define and store DBL at an RDA Server for later execution and drop stored DBL when it is not needed. The **R-DefineDBL** service element allows the client to define and the DB server to validate and store a specific DBL command. A DBL command has a statement, a command handle, optional argument and optional result formal parameters. The client supplies the statement and command handle. The parameters can be specified either by the client or the server. The command handle is valid until either it is dropped by R-DropDBL, or the referenced resource is closed by R-Close, or the dialogue is terminated permanently by negotiation. If more than one data resource is open, dataResourceHandle is necessary to differentiate them.

The **R-InvokeDBL** service element is used to execute an already stored command a fixed number of times and inform the client of its result.

The **R-DropDBL** service element is used to delete one or more defined DBL commands from the set of defined DBL Commands known to the RDA Server. One of the request parameters, list of commandHandle, specifies the command handles for commands which are to be deleted. If omitted, all currently defined commands will be deleted.

CHAPTER III

Protocol

An RDA server has to obey some server rules. Section 3.1 describes the Dialogue State Model which is composed of four entities. Section 3.2 shows the general server execution rules which can be applied to every service element. Section 3.2 shows how to apply these server rules to each individual service element.

3.1 Dialogue State Model

The **Dialogue State Model** is a data model showing the state of a dialogue between a client and an RDA server. It is made up of four entities: RDA dialogue entities, opened data resource entities, defined DBL entities and operation entities.

An **RDA dialogue entity** is created by a successful R-Initialize service requesting a new RDA dialogue. It exits until the dialogue is terminated. The RDA dialogue entity has the following attributes:

* dialogueID
  - dialogueIDClientInvocation
  - dialogueIDSuffix
* identityOfUser
* controlServicesAllowed
* controlAuthenticationData
* functionalUnitsNegotiated
* RDATransactionStatus
  - Transaction Not Open
  - Transaction Open
  - Transaction Terminating
* RDATransactionRolledBack

A **data resource** is a named object (or a collection of data and/or capabilities) which can be opened or closed. It is known to both the server and the client. The meaning of the data content and capabilities depend on the application of RDA which is determined by the RDA specialization. The client must open a data resource to gain an access to its data content or capability. RDA can support access to the nested directories and the other structured name spaces. Subordinate resources are grouped within their parent resource. A parent resource must be opened before the subordinate resources can be opened. DBL operations are also associated with data resources. More than one data resource can be opened at the same time. DBL operations can address data from more than one data resources if the DBL allows, but the DBL operation is defined in the context of one data resource only.

An **opened data resource entity** is created by a successful R-Open service and deleted by R-Close service. A data resource is opened within the context of dialogue, or within the context of another opened data resource, which is its parent data resource. An opened data resource entity may have only one parent and the the resource is automatically closed if its parent is closed. If the dialogue to which it belongs is terminated, then an opened data resource is also closed. The opened data resource entity has the following attributes:

* DialogueID
* dataResourceHandle
* parentDataResourceHandle
* dataResourceName

A **defined DBL entity** represents one DBL statement which the client uses in its invocation within a dialogue and is always related to an opened data resource entity. A defined DBL entity is created by a successful R-DefineDBL service request and deleted independently of opened data resource entities by the R-DropDBL service. The defined DBL entity has the following attributes:

* dialogueID
* commandHandle
* dataResourceHandle

Within the RDA server, every client's service request is modelled by **operation entity**. An operation entity lasts until the results or errors of the operation corresponding to that entity are returned. The following attributes comprise the operation entity:

* dialogueID
* operationID
* status
  - AWAITING EXECUTION
  - EXECUTING
  - FINISHED
  - CANCELLED
  - ABORTED
* cancelRequestReceived
* operationInvocationType
* operationArgument
* operationResult
* operationError

3.2 General Server Execution Rules

When a server receives a service indication primitive, it will create an operation entity with the initial values for a set of attributes. After the creation of the operation entity, if there already exists another operation entity with the same value for the dialogueID attribute and the same value for the operationID attribute, then the newly created operation entity will be updated as the following provided the value of its status attribute is AWAITING EXECUTION or EXECUTING:

* the attribute status will have the new value 'ABORTED'
* the attribute operationError will have the new value 'DuplicateOperationID'

Once an operation with a status attribute of AWAITING EXECUTION begins executing, its status is changed to EXECUTING. Any operation entity which has a status attribute value of EXECUTING must be updated according to the server execution rules for the particular operationInvocationType. The server execution rules for a particular service include the following three general types of rules.

* **Entity Modification Rules**: These specify the constraints on the manipulation of the entities and attributes which comprise the dialogue state. No updates are allowed except when explicitly allowed.
* **Result Rules**: These specify the constraints on the result responses or particular service request types.
* **Error Rules**: These define the conditions for specific errors to be returned. Such rules are prerequisite predicates for the errors. If no prerequisite predicates are specified for a particular error in a particular operation, then the issuance of error is never prevented for that operation. The error rules can also define the situations in which particular errors must occur. Such rules are the mandatory predicates for the errors. If no mandatory predicates are specified for a particular error in the context of a particular operation, then the issuance of error is never required for that operation.

When an operation ends abnormally, the status is set to FINISHED and the operationError is set to any error response value allowed. When an operation ends normally, the status is set to FINISHED and the operationError is set to any result response value satisfying the constraints.

An error or result response primitive must be generated only for an operation entity which has a status value of FINISHED, CANCELLED, or ABORTED. The operation entity must be deleted when the response primitive is issued. Response primitives must be issued in the following order.

* The R-Initialize response primitive will be issued before any other response on the dialogue.
* Following either an Error response to an R-BeginTransaction indication or TransactionRolledBack Error response to an R-ExecuteDBL or R-InvokeDBL indication the server must discard any later requests until it receives either an R-Rollback or R-Commit indication.
* Other than as stated above, response primitives of R-Cancel and R-Status must be issued earlier than the other primitives.
* Response primitives of all services other than R-Cancel and R-Status must be issued in the order in which the indication primitives were received.

The server can execute more than one operation at the same time. Also the server must ensure that the results are the same as the ones which have been executed serially.

When any implementor defined error occurs during AWAITING EXECUTION or EXECUTING, the status will have the new value 'ABORTED' and the operationError will have the new value 'OperationAborted' together with supplemental information.

After the server cancelled an executing operation to which R-Cancel service indication was issued, the status will have the new value 'CANCELLED' and the operationError will have the new value 'OperationCancelled'.

If the underlying association in use by a dialogue is failed, then all entities (dialogue, opened data resource, defined DBL, and operation) whose dialogueID attribute equals the dialogueID for the current dialogue must be deleted.

3.3 Server Rules for Each Service Element

When serving the **R-Initialize** service element, an RDA dialogue entity will be generated if an error is not returned. This RDA dialogue entity contains the following attributes: dialogueID, identityOfUser, controlServicesAllowed, controlAuthenticationData, functionalUnitsNegotiated, RDATransactionStatus, RDATransactionRolledBack. If a result is returned, the following result parameters must be satisfied with some constraints: controlServicesAllowed, controlAuthenticationData, functionalUnitsAllowed. One type of error, DuplicatedDialogueID exists in R-Initialize service element.

When serving the **R-Terminate** service element, the following entities will be deleted if an error is not returned:

* All RDA dialogue entities with a dialogueID attribute same as the current dialogue ID.
* All opened data resource entities with a dialogueID attribute same as the current dialogue ID.
* All defined DBL entities with a dialogueID attribute same as the current dialogue ID.

The Result Rules and additional Error Rules for this service element were not specified by International Standard. Table 3-1 shows the server rules for the Dialogue Management Services.

When serving the **R-BeginTransaction** service element, the attribute RDATransactionStatus of the dialogue entity will be set to the value RDATransactionOpen. There

are no result responses for this service element and additional error constraints for this service element were not specified by the International Standard.

Functional Unit Name	Service Elements	Types of Rules		
		Entity Modification	Result	Error
Dialogue Initialization Functional Unit	R-Initialize	dialogueID, identityOfUser, controlServicesAllowed, controlAuthenticationData, functionalUnitsNegotiated, RDATransactionStatus, RDATransactionRolledBack	controlServices-Allowed, control-AuthenticationData, functionalUnits-Allowed	Duplicate-DialogueID
Dialogue Termination Functional Unit	R-Terminate	All RDA dialogue, opened data resource, defined DBL entities with a dialogueID attribute same as the current dialogueID will be deleted.	Not Specified by International Standard	Not Specified by International Standard

Table 3-1 Server Rules for the Dialogue Management Services

When serving the **R-Commit** service element, the attribute RDATransactionStatus will have the new value, RDATransactionNotOpen. The RDATransactionRolledBack attribute is set to TRUE if the transaction has been aborted and all changes made to the DB have been rolled back. If a result is returned, then the result parameter transactionResult will have the value ROLLEDBACK if RDATransactionRolledBack is TRUE else this parameter will have the value COMMITTED. The additional Error Rules for this service element were not specified by the International Standard.

When serving the **R-Rollback** service element, the current dialogue entity will be updated as the follows. The attribute RDATransactionStatus will have the new value, RDATransactionNotOpen. The RDATransactionRolledBack attribute is set to TRUE. There are no Result Rules in R-Rollback service. The additional Error Rules for this service were not specified by the International Standard. Table 3-2 gives a summary of the server rules for handling the Transaction Management Functional Unit.

Functional Unit Name	Service Elements	Types of Rules		
		Entity Modification	Result	Error
Transaction Management Functional Unit	R-Begin-Transaction	RDATransactionStatus	No result responses	Not specified by International Standard
	R-Commit	RDATransactionStatus, RDATransactionRolledBack	transactionResult	Not specified by International Standard
	R-Rollback	RDATransactionStatus, RDATransactionRolledBack	No result rules	Not specified by International Standard

Table 3-2 Server Rules for the Transaction Management Services

There are 2 functional units in Control Services: Cancel Functional Unit and Status Functional Unit. The Cancel Functional Unit has **R-Cancel** service element. The target operation entities with the status attribute value AWAITING EXECUTION will be updated as the follows. The status attribute will have the value CANCELLED and the cancelRequestReceived attribute will have the value TRUE. The attribute operationError is set to OperationCancelled. The target operation entities with EXECUTING status will be set the attribute cancelRequestReceived as TRUE. Any R-Cancel result will be returned before any response is issued for an operation entity with TRUE cancelRequestReceived attribute value as the result of R-Cancel service. The Status Functional Unit contains the **R-Status** service element. There are no entity manipulation rules for R-Status service element. If a result is returned the following result parameters will satisfy the constraints: list of statusInformation, operationID, operationStatus, operationIDUnknown, awaitingExecution, executing, finished, cancelled, aborted. Table 3-3 summarizes the server rules for the Control Services.

Functional Unit Name	Service Elements	Types of Rules		
		Entity Modification	Result	Error
Cancel Functional Unit	R-Cancel	status, cancelRequestReceived, operationError (status attribute= AWAITING EXECUTION) cancelRequestReceived (status attribute = EXECUTING)	Any R-Cancel result will be returned before any response is issued for an operation entity with TRUE cancelRequestReceived as the result of R-Cancel service.	ControlAuthenti-cationFailure, DialogueIDUnknown
Status Functional Unit	R-Status	No entity manipulation rules.	list of statusInformation, operationID, operationStatus, operationIDUnknown, awaitingExecution executing, finished, cancelled, aborted	ControlAuthenti-cationFailure, DialogueIDUnknown

Table 3-3 Server Rules for the Control Services

In the Resource Handling Services, Resource Handling Functional Unit has R-Open and R-Close service elements. In serving the **R-Open** service element, an opened data resource entity will be generated if an error is not returned. This opened data resource entity contains the following attributes: dialogueID, dataResourceHandle, parentDataResourceHandle, dataResourceName. The Result Rules for this service were not specified by the International Standard.

When serving the **R-Close** service element, all opened data resource entities and defined DBL entities which were active will be deleted if an error was not returned. The following result parameters will be set by the constraints according to the given conditions: dataResourceHandle, dataResourceHandleUnknown, specificCloseException. The additional Error Rules for this service were not specified by the International Standard. Table 3-4 shows the server rules for the Resource Handling Services.

Functional Unit Name	Service Elements	Types of Rules		
		Entity Modification	Result	Error
Resource Handling Functional Unit	R-Open	dialogueID, dataResourceHandle, parentDataResourceHandle, dataResourceName	Not specified by International Standard.	ParentData-Resource-HandleUnknown, DataResource-Open, DuplicateData-ResourceHandle
	R-Close	All opened data resource entities and defined DBL entities which were active will be deleted if an error was not returned.	dataResource-Handle, dataResource-HandleUnknown, specificClose-Exception	Not specified by International Standard.

Table 3-4 Server Rules for the Resource Handling Services

There are 2 functional units in Database Language Services: Immediate Execution DBL Functional Unit and Stored Execution DBL Functional Unit. The Immediate Execution DBL Functional Unit has the **R-ExecuteDBL** service element. In the Entity Modification Rules, if an error occurs which causes the transaction to be rolled back, the RDATransactionRolledBack attribute of RDA dialogue entity for the current dialogue will have the new value TRUE. If a result is returned, the attribute, list of resultValues will have the number of entries which would be equal to the repetitionCount parameter on the R-ExecuteDBL indication primitive if singleArgument was specified. The number of entries in this parameter would be equal to the number of entries in the list of specificDBLArgument on the R-ExecuteDBL indication primitive if multipleArgument was specified. The number of entries in this parameter will be one if neither singleArgument nor multipleArgument was specified. There are 5 possible errors for this service: BadRepetitionCount, DataResourceHandleNotSpecified, DataResourceHanleUnknown, NoDataResourceAvailable, TransactionRolledBack.

The Stored Execution DBL Functional Unit has 3 service elements: R-DefineDBL, R-InvokeDBL, R-DropDBL. When serving the **R-DefineDBL** service element, a defined DBL entity will be generated with the following constraints on the initial attribute values if an error is not returned. The attribute dialogueID has the value of current dialogueID. The commandHandle attribute will have the commandHandle parameter value on the R-DefineDBL indication primitive. The attribute dataResourceHandle will have the dataResourceHandle parameter value on the R-DefineDBL indication primitive. The Result Rules for R-DefineDBL service element were not specified by the International Standard. There are 4 possible errors

for this service: DataResourceHandleNotSpecified, DataResourceHandleUnknown, DuplicateCommandHandle, NoDataResourceAvailable.

When serving the **R-InvokeDBL** service element, in the Entity Modification Rules, if an error which causes the transaction to be rolled back, then the RDA dialogue entity for the current dialogue will be modified as the attribute RDATransactionRolledBack having the new value TRUE. The result parameter, list of resultValues will satisfy the following constraints if a result is returned. If a result is returned, the attribute, list of resultValues will have the number of entries which would be equal to the repetitionCount parameter on the R-InvokeDBL indication primitive if singleArgument was specified. The number of entries in this parameter would be equal to the number of entries in the list of specificDBLArgument on the R-InvokeDBL indication primitive if multipleArgument was specified. The number of entries in this parameter will be one if neither singleArgument nor multipleArgument was specified.
Three possible errors for this service element are: BadRepetitionCount, CommandHandleUnknown, TransactionRolledBack.

When serving the **R-DropDBL** service element, all defined DBL entities which have a dialogueID attribute same as the current dialogue and whose commandHandle attribute is the same as an entry in the list of commandHandle parameter in the R-DropDBL indication primitive will be deleted if an error is not returned. The following result parameters will satisfy the constraints if a result is returned: commandHandle, commandHandleUnknown, specificDropDBLException. No Error Rules for R-DropDBL service element were specified by the International Standard. Table 3-5 summarizes the server rules for the DBL Services.

Functional Unit Name	Service Elements	Types of Rules		
		Entity Modification	Result	Error
Immediate Execution DBL FunctionalUnit	R-Execute-DBL	RDATransaction-RolledBack	list of result Values	BadRepetitionCount, DataResourceHandle-Unknown, DataResourceHandle-NotSpecified, NoDataResource-Available,Transac-tionRolledBack
Stored Execution DBL Functional Unit	R-Define-DBL	dialogueID, commandHandle, dataResourceHandle	Not specified by International Standard.	DataResourceHandle-Unknown, Data-ResourceHandleNot-Specified, DuplicateCommand-Handle, NoData-ResourceAvailable
	R-Invoke-DBL	RDATransaction-Rolledback	list of result Values	BadRepetitionCount, CommonHandle-Unknown, TransactionRolled-Back
	R-Drop-DBL	All defined DBL entities with a dialogueID equal the current dialogue and whose commandHanle is the same as an entry in the list of commandHandle parameter in the R-Drop-DBL indication will be deleted if an error is not returned.	commandHanle, command-HandleUnknown, specificDrop-DBLException	Not specified by International Standard.

Table 3-5 Server Rules for the DBL Services

CHAPTER IV

RDA Application Contexts

There are 2 RDA Application Contexts: RDA Basic Application Context and RDA TP Application Context (Figure 4-1). In these application contexts, the ACSE A-ASSOCIATE service is used to establish the association which is used by RDA. It is an RDA requirement that the AE title of the RDA client must have been communicated via the A-ASSOCIATE service. This requirement allows the AE title to be omitted from the R-Initialize service.

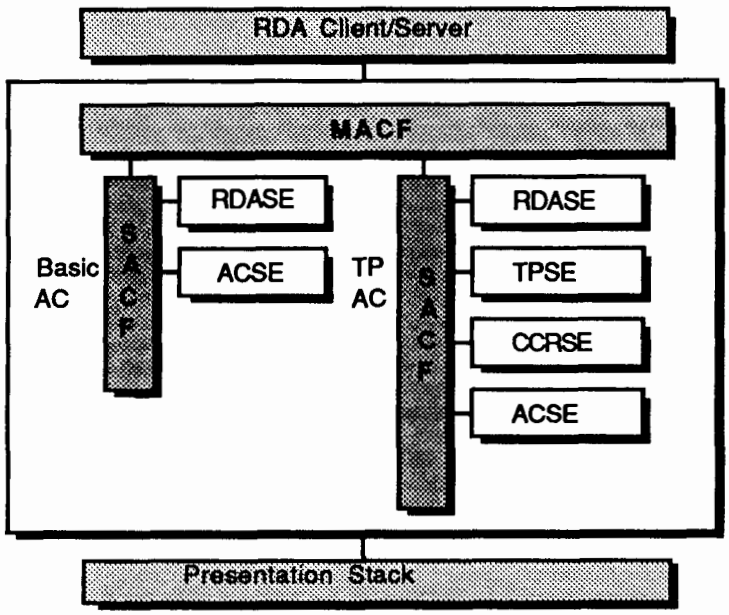


Figure 4-1 Two RDA Application Contexts

4.1 Basic Application Context

The **RDA Basic Application Context** uses the remote DB access facilities of RDA for DB transactions which access data by a single remote Open System and the association management facilities of ACSE. The ACSE service defines the operations for the establishment of an association and for the normal and abnormal release of an association. The RDA service

defines the operations for remote DB access including services for dialogue and transaction management. This application context imposes additional constraints on the use of ACSE services. This is the minimum application context which can perform RDA services. It is composed of the RDA ASE and ACSE. The Basic Application Context supports the A-ASSOCIATE, A-RELAESE, A-ABORT and A-P-ABORT service elements. The A-ASSOCIATE service element is used to establish an association and the other service elements are used to terminate an association. The A-RELEASE service element will not be used while a dialogue is active on an association.

4.1.1 SACF Rules

The SACF rules specified hereafter are in addition to rules already specified by the RDA ASE and ACSE. The Dialogue Termination Functional Unit is mandatory in the Basic Application Context.

1) Sequencing Rules

* A-Associate
  - An A-ASSOCIATE request must only be invoked by the RDA client.

* A-Release
  - An A-RELEASE request must not be invoked by either the RDA client or the RDA server while an RDA dialogue is active on the association.

* A-Abort
  - An RDA client that issues an A-ABORT request or that receives an A-ABORT indication for a particular association must delete all state information for the RDA dialogue active on that association.
  - An RDA server that issues an A-ABORT request or that receives an A-ABORT indication for a particular association must delete all state information, drop all defined DBL statements, and close all opened data resources for the RDA dialogue active on that association.

* A-P-Abort

- An RDA client that receives an A-P-ABORT indication for a particular association must delete all state information for the RDA dialogue active on that association.
- An RDA server that receives an A-ABORT indication for a particular association must delete all state information, drop all defined DBL statements, and close all opened data resources for the RDA dialogue active on that association.

2) Mapping Rules

* Mapping of ACSE APDUs
  - No additional rules are required.

* Mapping of RDA APDUs
  - All RDA APDUs map onto the user data parameter of the P-DATA request.

3) State Transition Diagrams

Figure 4-2 and Figure 4-3 show the state transitions of the RDA Basic Application Context.

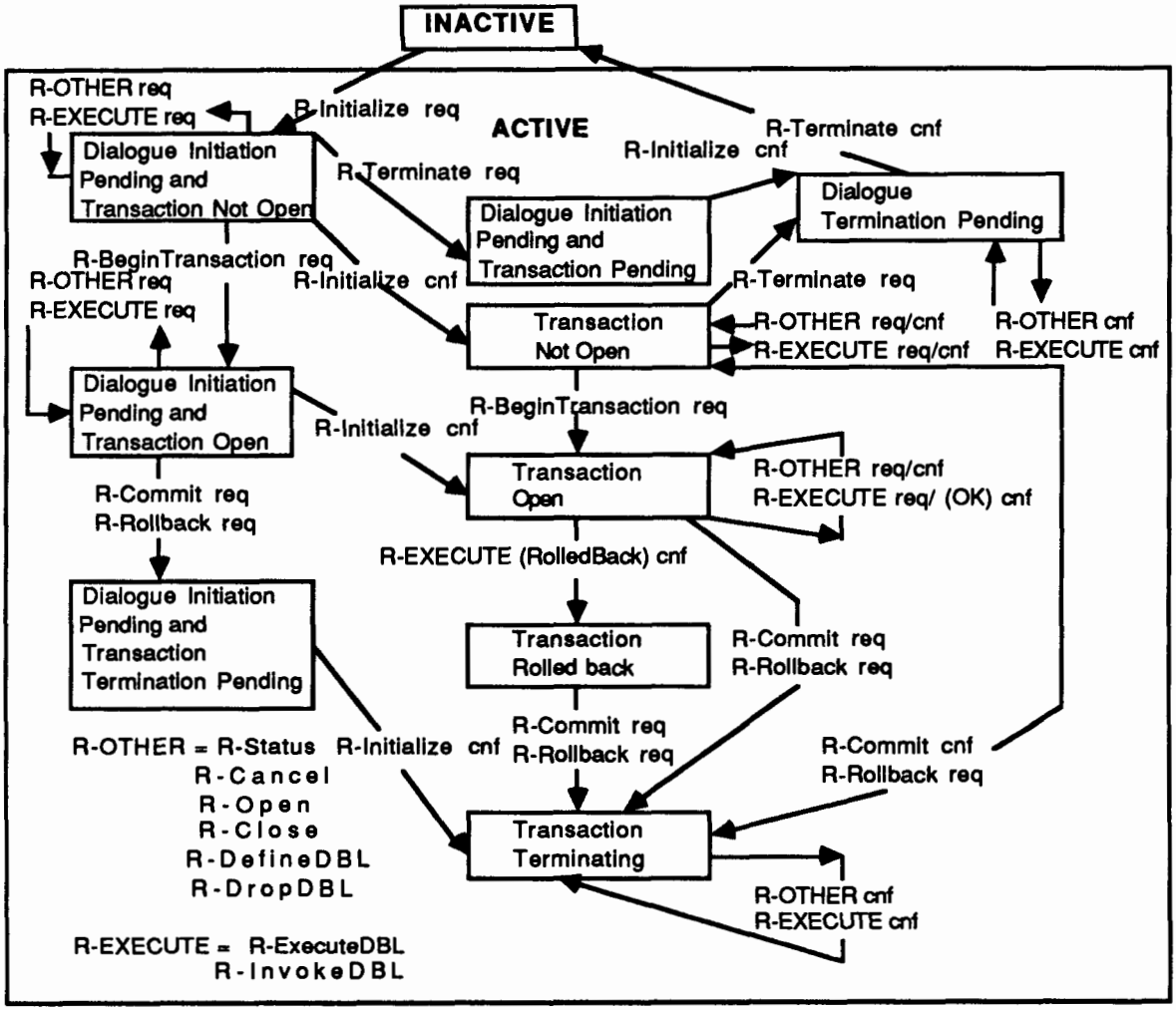


Figure 4-2 State Transition Diagram for RDA Basic Application Context - Client

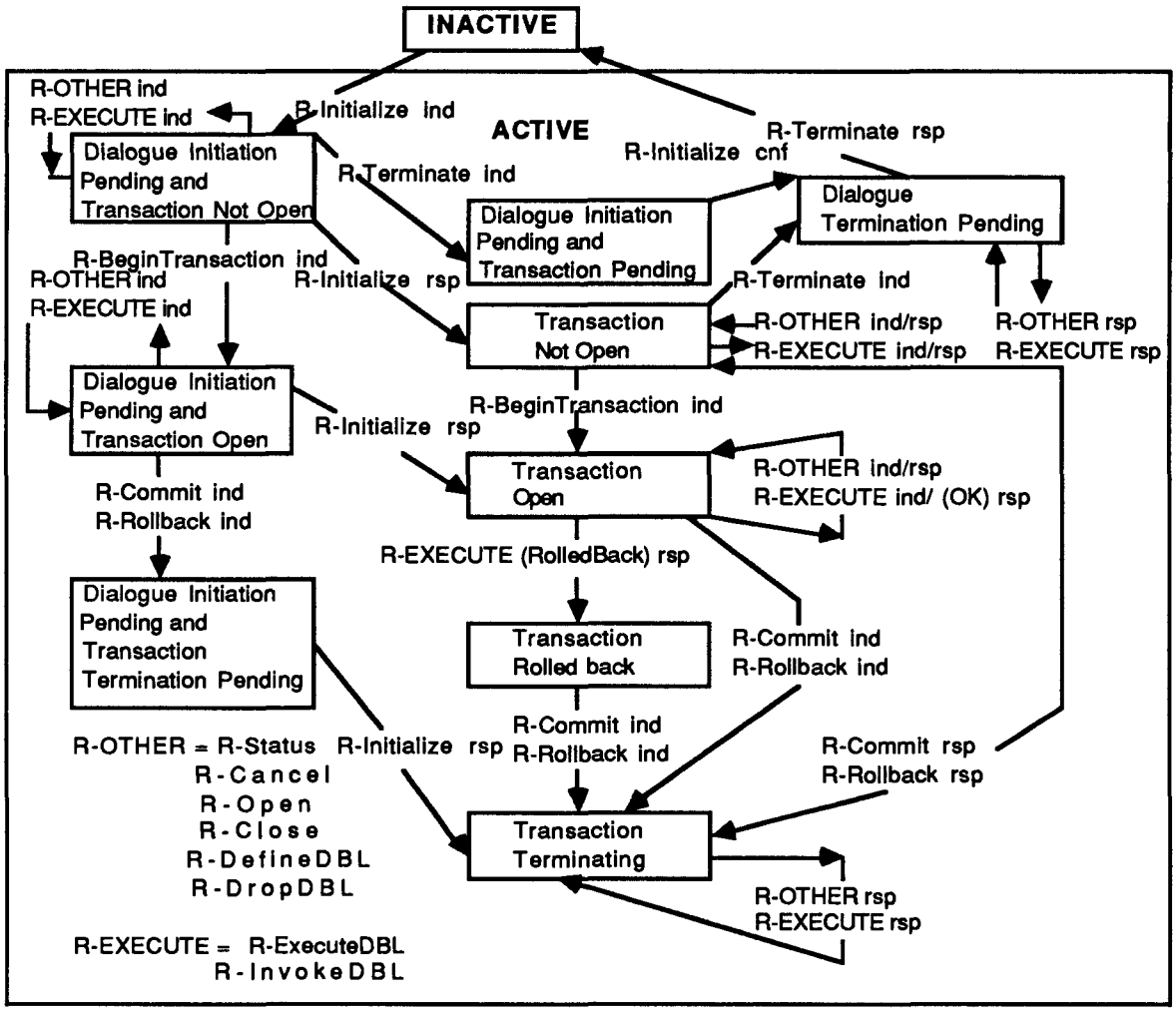


Figure 4-3 State Transition Diagram for RDA Basic Application Context - Server

4.2 TP Application Context

The **RDA TP Application Context** uses the dialogue management and transaction management facilities of TP and the control, resource handling, and DB language services of RDA for DB transactions which span more than one open system. In the RDA Application Context, the RDA client and server are each part of Transaction Processing Service User Invocation (TPSUI). The RDA service defines the operations for getting remote data access in the scope of one particular branch of a transaction tree. The TP service provides the transaction management facilities for the overall coordination of all local and remote resources in a reliable way to either successfully or unsuccessfully terminate the distributed transaction, achieving a consistent state of all resources. The TP service also includes the appropriate

recovery mechanisms to re-establish a consistent state of all resources following a communication or system failure. The TP service thus includes the necessary provisions to coordinate all remote services in order to ensure the atomicity, consistency, isolation, and durability characteristics of a distributed transaction. TP also provides the dialogue management facilities for the establishment, maintenance, and termination of dialogue. The RDA TP Application Context imposes additional constraints on both RDA and TP.

RDA-TP Application Context is composed of the ACSE, RDA ASE, TP ASE, and optionally CCR ASE.

4.2.1 SACF Rules

There are SACF rules in addition to rules already specified by the ASEs with the exception of rules implied by the following RDA services: R-BeginTransaction, R-Commit, R-Rollback and R-Terminate when the TP Chained Transactions Functional Unit is selected. The Dialogue Termination Functional Unit is mandatory when the TP Commit Functional Unit is not selected. The Dialogue Termination Functional Unit is optional when the TP Commit and Unchained Transactions Functional Units are selected. The Dialogue Termination Functional Unit would not be selected when the TP Commit and Chained Transactions Functional Units are selected.

1) Sequencing Rules

* RDA with TP Kernal Functional Unit

- TP-BEGIN-DIALOGUE, TP-P-REJECT, TP-U-REJECT:
A TP-BEGIN-DIALOGUE request will only be invoked by the client, and will only be followed by an R-Initialize request. Before an R-Initialize confirm has been received, the client cannot generate any TP service request. If the TP-BEGIN-DIALOGUE indication is rejected by either a TP-P-REJECT or a TP-U-REJECT indication, then the RDA dialogue entity will be deleted at the client. At this point of time, the dialogue entity does not yet exist at the server because the R-Initialize indication has not been responded to.

- TP-END-DIALOGUE:
If the dialogue establishment fails, the RDA server will generate a TP-END-DIALOGUE request. When the server generates an R-Terminate response which

includes a normal result, the server will then also generate a TP-END-DIALOGUE request. A TP-END-DIALOGUE request will only be preceded by an R-Initialize response which includes an error, or an R-Terminate response which includes a normal result.

- TP-DATA:
All the TP sequencing rules which apply to the transfer of application(TP-DATA service) apply to the following RDA service elements: R-Initialize, R-Terminate, R-Cancel, R-Status, R-Open, R-Close, R-ExecuteDBL, R-DefineDBL, R-InvokeDBL, and R-DropDBL.

- TP-U-ERROR:
A client that generates a TP-U-ERROR request deletes all operation entities associated with the RDA dialogue. A server that receives a TP-U-ERROR indication deletes all operation entities associated with the dialogue. The server TPSUI also deletes all opened data resource entities and defined DBL entities associated the deleted operation entities. A server that generates a TP-U-ERROR request deletes all operation entities associated with the dialogue. The server TPSUI also deletes all opened data resource entities and defined DBL entities associated with the deleted operation entities. A client that receives a TP-U-ERROR indication deletes all operation entities associated with the dialogue.

- TP-U-ABORT:
A TPSUI which generates a TP-U-ABORT request for a particular TP dialogue will delete the RDA dialogue entity for that TP dialogue. A TPSUI which receives a TP-U-ABORT indication for a particular TP dialogue will delete the dialogue entity for that dialogue.

- TP-P-ABORT:
A TPSUI which receives a TP-P-ABORT indication for a particular TP dialogue will delete the dialogue entity for that dialogue.

* RDA with Polarized Control Functional Unit

- TP-GRANT-CONTROL:

The RDA client must issue a TP-GRANT-CONTROL request to surrender control to the RDA server before the RDA server can respond to any outstanding RDA client request. The RDA server must generate TP-GRANT-CONTROL request to surrender control to the client before the client can issue additional requests.

* RDA with TP Commit Functional Unit

- TP-PREPARE:
The Data-permitted parameter must be present on the TP-PREPARE request when issued by the client while operation entities are still outstanding.

- TP-ROLLBACK:
A client that generates a TP-ROLLBACK request deletes all operation entities associated with the dialogue. A server that receives a TP-ROLLBACK indication deletes all operation entities associated with the dialogue. The server TPSUI also deletes all opened data resource entities and defined DBL entities associated with the deleted operation entities.

- TP-COMMIT-COMPLETE:
An RDA client that receives a TP-COMMIT-COMPLETE indication after it has previously issued a TP-DEFERRED-END-DIALOGUE request must delete all state information for the dialogue. The RDAPM returns to state I. An RDA client that receives a TP-COMMIT-COMPLETE indication after it has previously received a TP-DEFERRED-END-DIALOGUE indication must delete all state information for the dialogue. The RDAPM returns to state I.

* RDA with TP Unchained Transactions Functional Unit

- TP-COMMIT-COMPLETE: The rules are the same as the rules for TP-COMMIT-COMPLETE of the RDA with TP Commit Functional Unit.

2) Mapping Rules

* Mapping of TP APDUs
  - No additional rules are required.

* Mapping of RDA APDUs
  - All RDA APDUs map onto the user data parameter of the P-DATA request, except those that are concatenated with TP APDUs or CCR APDUs that have alternate mappings.

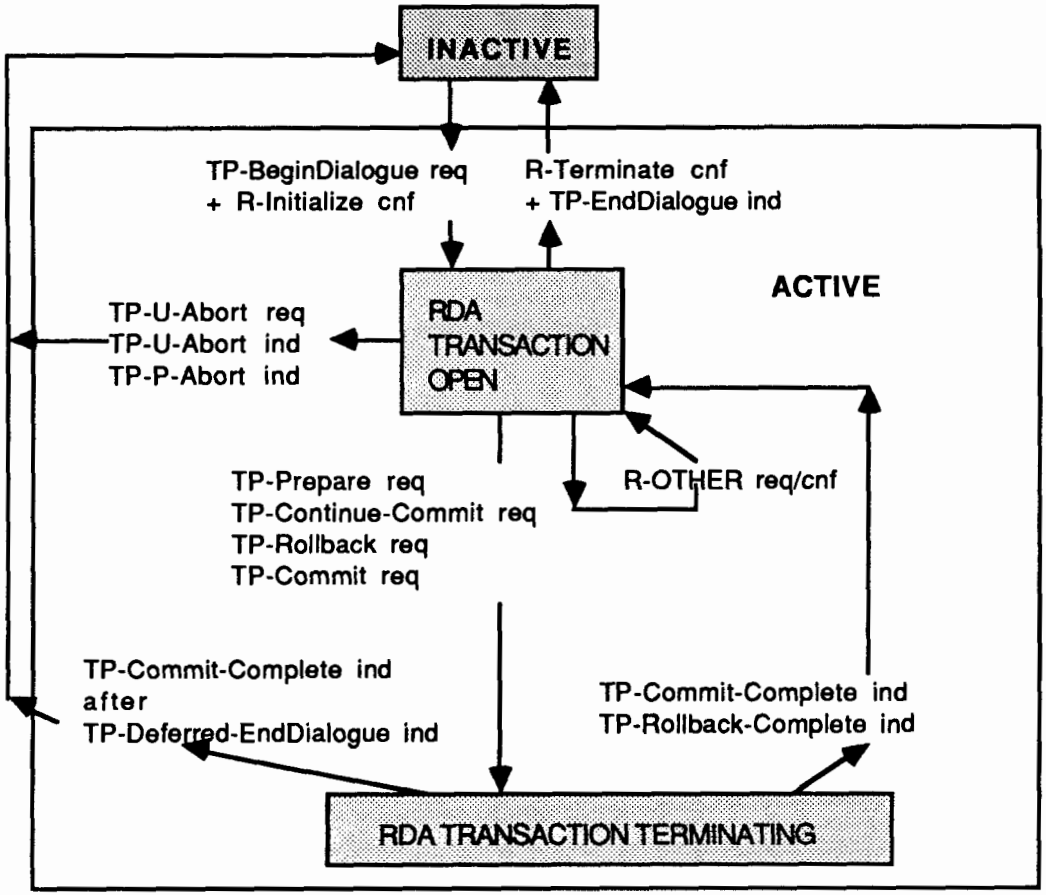
3) Concatenation Rules

* RDA with TP Kernal Functional Unit
  - TP-BEGIN-DIALOGUE: The R-Initialize-RI APDU is concatenated to the TP-BEGIN-DIALOGUE-RI APDU.
  - TP-END-DIALOGUE: The TP-END-DIALOGUE-RI APDU is concatenated to the R-Initialize-RC APDU that contains a result. The TP-END-DIALOGUE-RI APDU is concatenated to the R-Terminate-RC APDU that contains a result.

* RDA with Other TP Functional Units
  - No additional rules are required.

4) State Transition Diagrams

Figures 4-4, 4-5, 4-6, 4-7 show the state transitions of the RDA TP Application Context.



R-OTHER = R-Status
R-Cancel
R-Open
R-Close
R-ExecuteDBL
R-DefineDBL
R-InvokeDBL
R-DropDBL

Figure 4-4 State Trnsition Diagram for TP Application Context - Client (Chained Transactions)

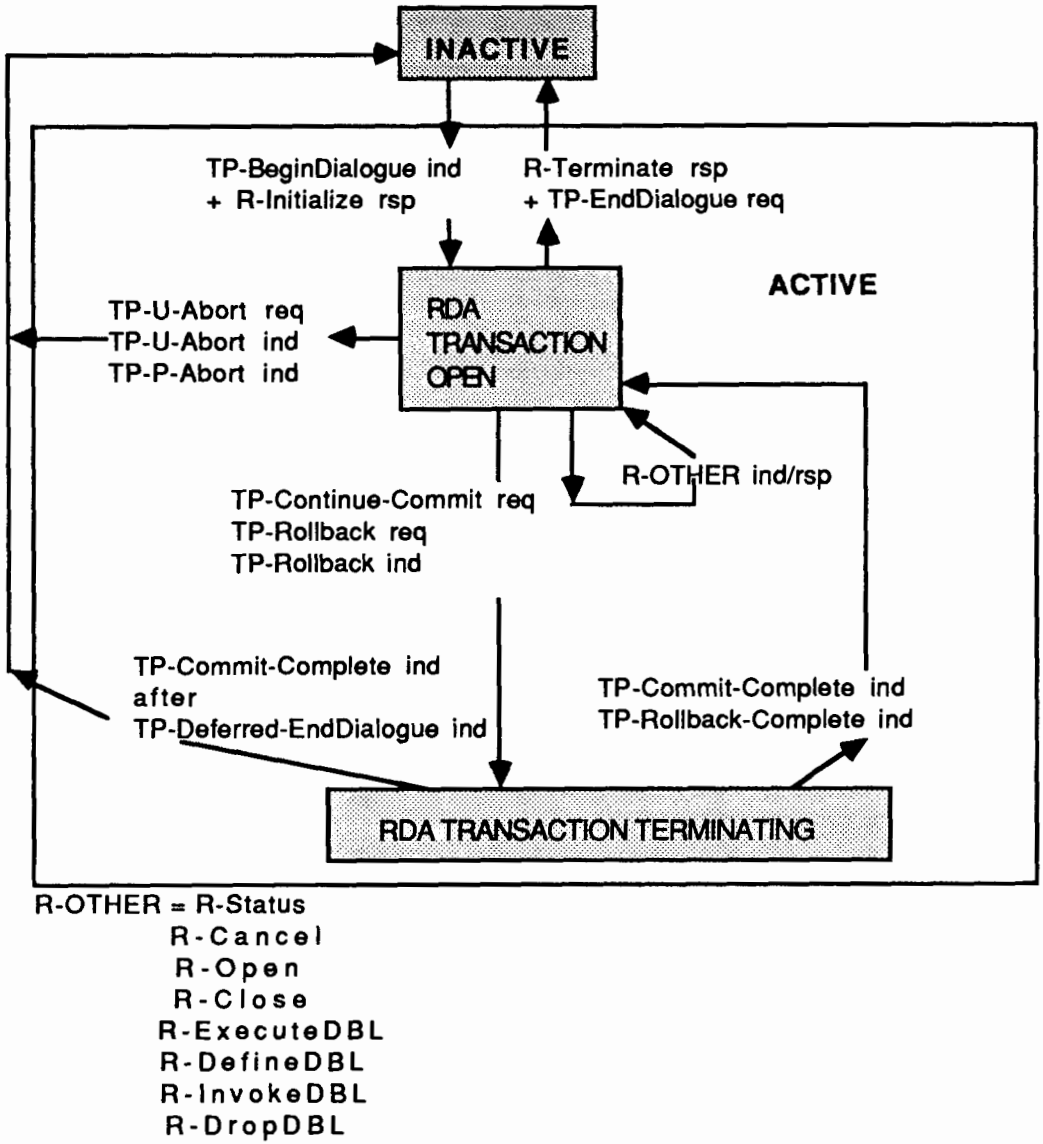
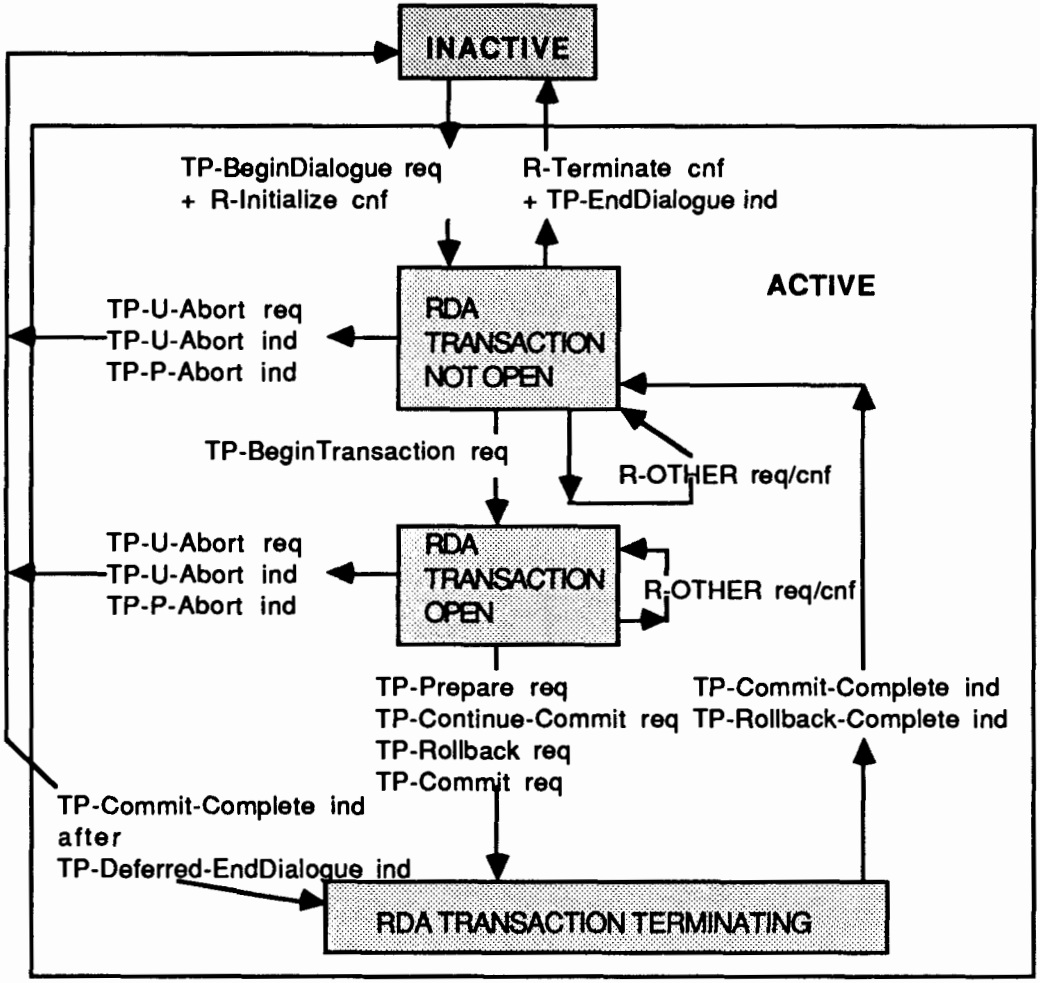
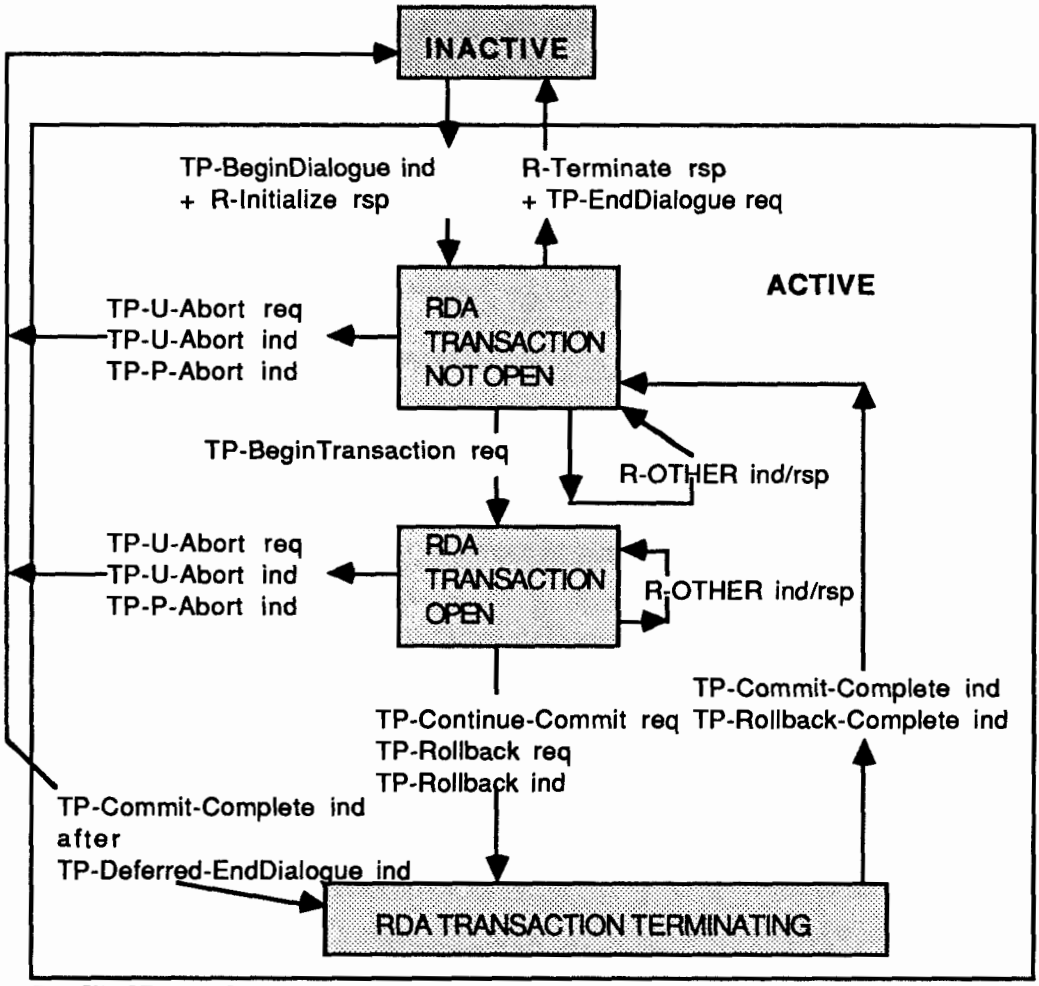


Figure 4-5 State Trnsition Diagram for TP Application Context - Server (Chained Transactions)



R-OTHER = R-Status
R-Cancel
R-Open
R-Close
R-ExecuteDBL
R-DefineDBL
R-InvokeDBL
R-DropDBL

Figure 4-6 State Trnsition Diagram for TP Application Context - Client (Unchained Transactions)



R-OTHER = R-Status
R-Cancel
R-Open
R-Close
R-ExecuteDBL
R-DefineDBL
R-InvokeDBL
R-DropDBL

Figure 4-7 State Trnsition Diagram for TP Application Context - Server (Unchained Transactions)

CHAPTER V

Specialization

The general principle of **specialization** is that the specific standard will add to the provisions of the RDA Generic standard. It is not allowed for the specialization to replace the provisions by different provisions. RDA specialization defines the characteristics and capabilities of RDA data resources, the characteristics and semantics of RDA DBL commands. To be complete, the definition must state the effect of each permissible operation both on the DB and the result returned, any required sequencing rules containing the execution order, and any required restrictions on execution in or outside a transaction.

For each **service** the RDA specialization must define any required constraints on the permissible values of parameters. If a parameter is specialization defined, then the RDA specialization must either specify the permissible values and their meanings, including whether the parameter is mandatory, optional, or conditional or state that the parameter is not used. The RDA specialization must define any required additional service sequencing rules.

The RDA specialization must define any additional **Dialogue State Model** entities, and their attributes and any additional attributes of the entities in the RDA Generic standard's Dialogue State Model. For each **server operation**, the RDA specialization must define any additional:

* constraints on the permissible values of parameters.
* constraints to be satisfied in order that a result be returned.
* conditions that permit or require an error to be returned.
* Entity Manipulation Rules.
* Result Rules.

If a parameter is specialization defined, the RDA specialization must either specify the permissible values and their meanings, or state that the parameter is not used.

The formal definition of the specific **RDA protocol** can be specified in an ASN.1 module that follows the content, structure, and rules specified in the Protocol Data Units in the Appendix. The module must specify the type definitions for those parameters listed as being untyped in the RDA Generic standard. Those untyped parameters which are not required by the

RDA Specialization must be excluded from the specialization module. It is also the responsibility of each RDA specialization to assign an OBJECT IDENTIFIER to the abstract syntax for use in Presentation service.

The RDA specialization will define the **application contexts** and any additional constraints and sequencing rules.

An RDA Specialization will also include **conformance** to the conformance rules.

Section 5.1 describes RDA Specialization concept. In Section 5.2, SQL and its Specialization is introduced. Section 5.3 and 5.4 shows RDA SQL specialization services and RDA SQL specialization data types respectively. Section 5.5 discusses about the protocol.

5.1 Introduction

SQL, formerly known as SEQUEL, is a relational DB language whose name is derived from Structured Query Language which was designed and implemented originally as the interface for an experimental relational DB system, System R. It in now used in a lot of commercial DB systems and the entire DB system is sold under the same name SQL. The followings are the major characteristics of SQL:

- SQL supports a simple data structure which is a table.
- SQL supports the relational algebra operators of PROJECT, SELECT, and JOIN. Also, it operates on all relations to produce new relations.
- Table creation, querying and modifying, and view definition can be combined into a uniform syntax.
- SQL can be used as a stand-alone query language and within a programming language by embedding it in a host language.
- Physical and logical data independence can be provided to a large extent.

SQL is a DBL which has a lot of facilities. It is both Data Definition Language (DDL) and Data Manipulation Language (DML). So, it has the statements for data definitions, updates, and queries. Additionally, it has the view definition facilities on the DB, the creation and drop facility of indexes on the files which represent relations, the embedding facility of SQL statements into a high-level programming languages like Pascal, C or PL/I.

The RDA SQL Specialization complements the Generic RDA in order to define:

* the capabilities of an SQL DB server supporting dialogues with clients.
* a model of dialogues between the SQL DB server and remote users.
* a model of dialogues between an RDA client and SQL server.
* an abstract service interface for the RDA SQL ASE, which models the communications facilities supporting interaction between the SQL client and SQL server.
* the RDA SQL ASE protocol to support the RDA SQL service.
* the characteristics of application contexts which include the RDA SQL ASE.
* the application contexts which support remote DB access using SQL:
  - RDA Basic Application Context
  - RDA TP Application Context

The Specialization standard does not specify individual implementations or products, nor does it constrain the implementation of entities and interfaces in the computer system. Also, it does not define a programmatic interface.

The RDA SQL Specialization is formally defined in an ASN Module that is derived from the Specialization module template given in the Appendix. The RDA SQL Specialization module provides definitions for those types listed in the template as being undefined by the Generic standard. In that module, the Generic RDA data types named Specificxxx... are renamed to SQLxxx....

The RDA SQL Specialization defines a means of communicating SQL DBL statements and their parameters from an RDA client to an RDA server, and of returning the results of those statements. Database Language SQL is supported at various levels of conformance, determined by the document, ISO/IEC DIS 9075:199x(E). When the Conformance Level specifies a year value of 1987 or 1989, ISO/IEC 9075:1989(E) will be referenced. In case of a year value of 199x, the relevant Draft International Standard is ISO/IEC DIS 9075:199x(E).

5.2 RDA SQL Specialization Service Model

5.2.1 Mapping to the General Model of the RDA Service

The RDA server that is the object of RDA services must conform to the SQL DB model defined in ISO/IEC 9075 (Database Language SQL) is called the **SQL Server**. The Generic RDA entity Data Resource is an SQL DB as defined in ISO/IEC 9075 is called as the **SQL Database Resource.**

The R-Open service causes an SQL Database Resource at the SQL Server to be made accessible to the RDA client in requests for Database Language Services. The dataResourcename in R-Open is the SQL DB name known to the server. There are no hierarchical data resources in the RDA SQL Specialization. Nested R-Open's are not supported, and parentDataResourcehandle cannot be supplied as a parameter to R-Open.

The R-Close service makes an SQL Database Resource at the SQL server to be made inaccessible to the RDA client in requests for Database Language Services.

An implementor must provide an SQL server at which one or multiple SQL Database Resources are available. The client must perform an R-Close operation before invoking another R-Open so that only a single SQL Database Resource is available at a time.

The **RDA Dialogue State Model** is as defined in ISO/IEC DIS 9579-1 (RDA Part 1: Generic) with some additional attributes:

* To the RDA Dialogue Entity
  - **sQLConformanceLevel**: This parameter shows the ISO/SQL standard for the DBL statements to be used in the RDA DBL operations. The Object Identifiers are defined in ISO/IEC DIS 9075:199x(E).
  - **userData**: This is defined by the SQL server implementor.

* To the Opened Data Resource Entity
  - **SQLAccessControl**: This parameter value is supplied by the client to authenticate the right to open the SQL DB resources for the required usage.
  - **SQLUsageMode**: This parameter specifies the access mode to the SQL DB resource. If 'retrieval' mode is selected, all the objects within that resource can be accessed for read only purposes and any attempt to update any object within that resource will generate an error usageModeViolation. If 'update' mode is selected then update, insert, delete, and drop of the objects within that resource are permitted. The applicable privileges may further restrict the access to any of the objects within that resource irrespective of the SQLUsageMode. The default value for SQLUsageMode is 'retrieval'.

* To the Defined DBL Entity
  - **SQLDBLArgumentSpecification**: as defined in "RDA SQL DBL Argument and Results"
  - **SQLDBLResultSpecification**: as defined in "RDA SQL DBL Argument and Results"

- **SQLDBLStatement:** as defined in "RDA SQL DBL Operations"

5.2.2 Mapping to the Concepts of Database Language SQL

This section relates the relevant concepts defined in the ISO/IEC 9075 (Database Language SQL) to the DB model contained in the RDA server. SQL statements are executed by the DBMS at the SQL server exactly as if they were embedded in a host program local to the DBMS. Any exception or completion code generated by the DBMS is always returned to the RDA client. All responsibility for defining a collating sequence resides with the SQL server only.

5.3 RDA SQL Specialization Service

This section describes the expansion of the Generic RDA service parameters which are Specialization defined. These are the additional specifications to the ISO/IEC DIS 9579-1 (RDA Part 1: Generic). Table 5-1 shows the RDA services for the SQL specialization.

Service Name	Functional Units	Service Elements
Dialogue Managenent	Dialogue Initialization	R-Initialize
	Dialogue Termination	R-Terminate
Transaction Management	Transaction Management	R-BeginTransaction R-Commit R-Rollback
Control	Cancel	R-Cancel
	Status	R-Status
Resource Handling	Resource Handling	R-Open R-Close
Database Language	Immediate Execution DBL	R-ExecuteDBL
	Stored Execution DBL	R-DefineDBL R-InvokeDBL R-DropDBL

Table 5-1 RDA Services for the SQL Specialization

5.3.1 Dialogue Management Services

*** Dialogue Initialization Functional Unit**

- R-Initialize Service

  Invocation parameters

  SQLInitializeArgument: This parameter is used to negotiate the level of support desired by the RDA client.

  sQLConformanceLevel: This identifies the ISO/SQL standard for the DBL statements to be used in the RDA DBL operations. The Object Identifiers are defined in the ISO/IEC DIS 9075:199x(E).

  userData: This is defined by the implementor of the SQL server.

  Result parameters

  SQLInitializeResult: This is used by the SQL server to report the support it can provide when it cannot meet the support requested by the RDA client.

  sQLConformanceLevel: This identifies the ISO/SQL standard for the DBL statements to be used in the RDA DBL operations. The Object Identifiers are defined in the ISO/IEC DIS 9075:199x(E).

  userData: This is defined by the implementor of the SQL server.

  Error parameters

  SQLInitializeError: This is used by the SQL server to report errors.

  invalidSQLConformanceLevel: The value of sQLConformanceLevel is not allowed.

*** Dialogue Termination Functional Unit**

None of the specific invocations, results or errors for this service are used in this Specialization.

5.3.2 Transaction Management Services

*** Transaction Management Functional Unit**

All the following services have no specific invocation, result or error parameters.

- R-BeginTransaction Service
- R-Commit Service
- R-Rollback Service

5.3.3 Control Services

All the following services for the following two functional units have no specific invocation, result or error parameters.

*** Cancel Functional Unit**

- R-Cancel service

*** Status Functional Unit**

- R-Status Service

5.3.4 Resource Handling Services

*** Resource Handling Functional Unit**

- R-Open Service

  Invocation parameters

  SQLOpenArgument: This parameter contains information sent by the SQL client whose meaning is specific to SQL.

  charSet: This is an Object Identifier that uniquely identifies the specification of a coded character set. ISO 8824:1989(E) (ASN.1) specifies the description of the assignment of Object identifier values. The character set identified by that specification must be used by the client as the default in this RDA dialogue for all character data

arguments that are associated with DBL statements accessing the data resource opened by this service.

SQLAccessControl

SQLUsageMode

Result parameters

SQLOpenResult: This parameter contains information returned by the SQL server whose meaning is specific to SQL.

charSet

charSetNotSupported: The character set declared by the client is not supported by the server.

Error parameters

SQLOpenError: This parameter is used by the SQL server to report errors.

sQLAccessControlViolation: The value of SQLAccessControl is not allowed.

sQLUsageModeViolation: The value of SQLUsageMode is not allowed.

sQLDatabaseresourceAlreadyOpenError: The multiple open of multiple SQL DB resources is not allowed.

- R-Close Service

No specific invocation, results or errors for this service are used in this Specialization.

5.3.5 Database Language Services

*** Immediate Execution DBL Functional Unit**

- R-ExecuteDBL Service

Invocation parameters

SQLDBLStatement

SQLDBLArgumentSpecification

SQLDBLResultSpecification

SQLDBLArgumentValues

Result parameters

SQLDBLResultSpecification

ResultValues

SQLDBLException

SQLDBLResultvalues

Error parameters

SQLExcuteDBLError: This parameter is used by the SQL server to report errors.

sQLUsageModeViolation: The value of SQLUsageMode is in conflict with this request.

sQLDBLTransactionStatementNotAllowed: The content of SQLDBLStatement performs transaction management (e.g., SQL <commit statement> or SQL <rollback statement>) which is not permitted by the RDA SQL specialization.

sQLDBLArgumentCountMismatch: The number of entries in sQLDBLArgumentValues parameter is not the same as in its associated sQLDBLArgumentSpecification parameter.

sQLDBLArgumentTypeMismatch: The type of entries in SQLDBLArgumentValues parameter is not the same as in its associated sQLDBLArgumentspecification parameter.

hostIdentifierError: The SQL server detected an error in an SQL variable name in an SQL statement.

rDATransactionNotOpen: No transaction is open and an attempt was made to execute an operation containing an SQL Data statement.

*** Stored Execution DBL Functional Unit**

- R-DefineDBL Service

Invocation parameters

SQLDBLStatement

SQLDBLArgumentSpecification

SQLDBLResultSpecification

Result parameters

SQLDBLResultSpecification

SQLDBLException

Error parameters

SQLDefineDBLError: This parameter is used by the SQL server to report errors.

sQLUsageModeViolation: The value of SQLUsageMode is in conflict with this request.

sQLDBLTransactionStatementNotAllowed: The content of SQLDBLStatement performs transaction management (e.g., SQL <commit statement> or SQL <rollback statement>) which is not permitted by the RDA SQL specialization.

hostIdentifierError: The SQL server detected an error in an SQL variable name in an SQL statement.

- R-InvokeDBL Service

Invocation parameters

SQLDBLArgumentValues

Result parameters

ResultValues

SQLDBLException

SQLDBLResultValues

SQLDBLResultSpecification

Error parameters

SQLInvokeDBLError: This parameter is used by the SQL server to report errors.

sQLUsageModeViolation: The value of SQLUsageMode is in conflict with this request.

sQLDBLArgumentCountMismatch: The number of entries in sQLDBLArgumentValues parameter is

not the same as in its associated sQLDBLArgumentSpecification parameter.

sQLDBLArgumentTypeMismatch: The type of entries in SQLDBLArgumentValues parameter is not the same as in its associated sQLDBLArgumentspecification parameter.

rDATransactionNotOpen: No transaction is open and an attempt was made to execute an operation containing an SQL Data statement.

- R-DropDBL Service

  No specific invocation, results or errors for this service are used in this Specialization.

5.3.6 Sequencing Rules

There are no additional SQL specific sequencing rules.

5.4 RDA SQL Specialization Data Types

5.4.1 RDA SQL DBL Operations

SQL Database Language Operations are specified by ASN.1 notation. An SQL statement is communicated as a character string. The ASN.1 data type SQLDBLStatement can be defined as follows:

```
SQLDBLStatement        ::=    CHOICE
    {  sqlDataStatement       [0] IMPLICIT SQLDataStatement,
       sqlSchemaStatement     [1] IMPLICIT SQLSchemaStatement
       }

SQLDataStatement       ::=    IA5String

SQLSchemaStatement     ::=    IA5String
```

The SQL server is responsible for ensuring that the content of SQLDBLStatement is permitted by an implementation claiming conformance to ISO/IEC 9075 (Database Language SQL) at the level of conformance established for the current RDA dialogue and may return an sQLCode or sQLState value, as appropriate to the SQL conformance level, if the statement in non-conforming. Additionally, if the content of SQLDBLStatement performs transaction management (e.g., SQL <commit statement> or SQL ,rollback statement>), then the SQL server must return the error sQLDBLTransactionStetementNotAllowed.

5.4.2 RDA SQL DBL Arguments and Results

How can we define the formal and actual parameters for DBL SQL statements? The Specific data types for DBL SQL statement parameters can be defined as the followings:

```
SQLDBLArgumentSpecification   ::=   SEQUENCE OF SQLDataTypeDescriptor
   -- defined in the section "RDA SQL DBL Descriptors"

SQLDBLResulttSpecification    ::=   SEQUENCE OF SQLDataTypeDescriptor
   -- defined in the section "RDA SQL DBL Descriptors"

SQLDBLArgumentValues  ::=   SQLValueList

SQLResultValues  ::=   SQLValueList

SQLValueList      ::=   CHOICE
   {  external          EXTERNAL,
      internal          SEQUENCE OF SQLValue
   }

SQLValue          ::=   SEQUENCE
   {  dataItem          ::= CHOICE
      {  characterItem        [0] IMPLICIT OCTET STRING,
         numericItem          [1] IMPLICIT INTEGER,
         decimalItem          [2] IMPLICIT INTEGER,
         integerItem          [3] IMPLICIT INTEGER,
         smallIntItem         [4] IMPLICIT INTEGER,
         floatItem            [5] IMPLICIT REAL,
```

```
        realItem              [6] IMPLICIT REAL,
        doublePrecisionItem   [7] IMPLICIT REAL,
        }                     OPTIONAL,
    indicator                 [30] IMPLICIT INTEGER OPTIONAL
    }
```

The meaning of these types can be defined as the following:

SQLDBLArgumentSpecification: This data type is composed of a sequence of SQL data type descriptors. Each SQL data type descriptor defines a separate occurrence of an input <variable specification> in the DBL statement. The nth element of the sequence defines the nth occurrence of an input <variable specification> in the DBL statement.

SQLDBLResultSpecification: This data type is composed of a sequence of SQL data type descriptors. Each SQL data type descriptor defines a separate occurrence of an input <variable specification> in the DBL statement. The nth element of the sequence defines the nth occurrence of an output <variable specification> in the DBL statement.

SQLDBLArgumentValues: This data type carries the values of the argument parameters defined in the SQLDBLArgumentSpecification. The type is defined as either a sequence of individual Data Values and associated indicators, or as EXTERNAL. The latter permits for the option of defining a separate abstract syntax for SQL arguments.

SQLDBLResultValues: This data type carries the values of the result parameters defined in the SQLDBLResultSpecification. The type is defined as either a sequence of individual Data Values and associated indicators, or as EXTERNAL. The latter permits for the option of defining a separate abstract syntax for SQL results.

A <variable specification> may contain either one <embedded variable name>, if no <indicator variable> is included, or two <embedded variable name>s, if an <indicator variable> is included. The first or only <embedded variable name> will be called as the Data variable, the second as the Indicator variable.

An SQL statement may contain references to parameters in the Argument and/or the result. Each parameter reference in the <SQL statement> must be replaced in the SQLDBLStatement by the <embedded variable name> ":H".

The figure 5-1 given below shows how the actual data is referenced:

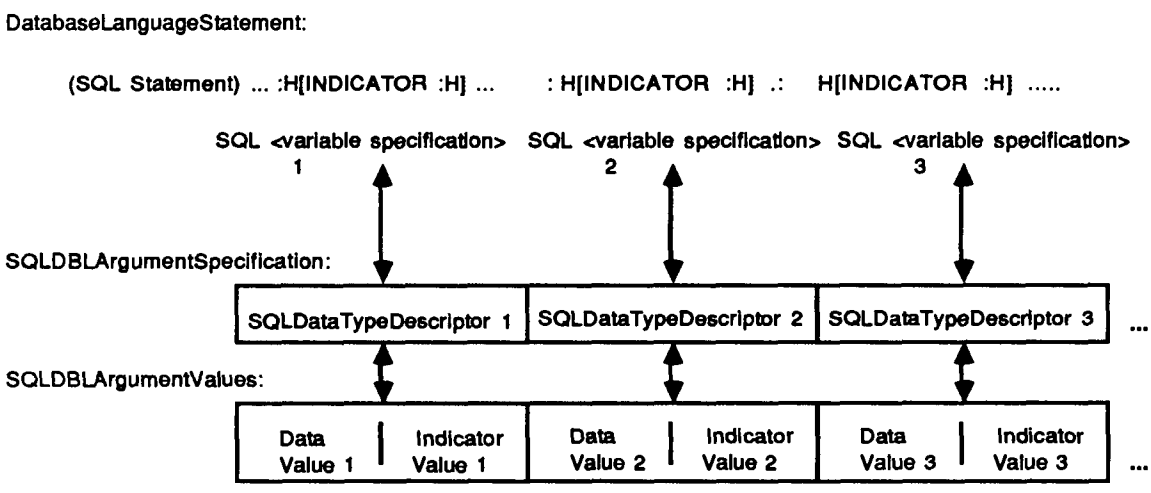


Figure 5-1 Referencing of the Embedded SQL Variables

Each <variable specification> corresponds to exactly 2 values, the first being the value of the Data Variable (Data Value in the figure) and the second being the value of the Indicator Variable (Indicator Value in the figure). If the Data Value is null, then it must be omitted and the Indicator Value must be included. If the Indicator Value is zero, then it must be omitted and the Data Value will be included.

5.4.3 RDA SQL DBL Descriptors

```
SQLDataTypeDescriptor  ::= SEQUENCE
    {  colName                  [1] IMPLICIT VisibleString OPTIONAL,
       varName                  [2] IMPLICIT IA5String OPTIONAL,
       indName                  [3] IMPLICIT IA5String OPTIONAL,
       typeDescriptor           CHOICE
       {  characterType         [5] IMPLICIT SEQUENCE
          - -   SQL Type: character
          {     charSet                 OBJECT IDENTIFIER OPTIONAL,
                length                  INTEGER,
                fixedLengthEncoding     BOOLEAN
```

```
        },
numericType             [6] IMPLICIT SEQUENCE
- -     SQL Type: numeric
{       precision               INTEGER,
        scale                   INTEGER
        },
decimalType             [7] IMPLICIT SEQUENCE
-- SQL Type: decimal
{       precision               INTEGER,
        scale                   INTEGER
        },
integerType             [8] IMPLICIT SEQUENCE
-- SQL Type: integer
{       precision               INTEGER,
        precisionBase           ENUMERATED
        {       binary                  (0),
                decimal                 (1)
                }
        },
smallIntType            [9] IMPLICIT SEQUENCE
-- SQL Type: smallInt
{       precision               INTEGER,
        precisionBase           ENUMERATED
        {       binary                  (0),
                decimal                 (1)
                }
        },
floatType               [10] IMPLICIT SEQUENCE
-- SQL Type: float
{       mantissaPrecision       INTEGER,
        exponentPrecision       INTEGER
        },
realType                [11] IMPLICIT SEQUENCE
-- SQL type: real
{       mantissaPrecision       INTEGER,
        exponentPrecision       INTEGER
```

```
            },
        doublePrecisionType  [12] IMPLICIT SEQUENCE
        -- SQL Type: doublePrecision
        {       mantissaPrecision     INTEGER,
                exponentPrecision     INTEGER
                }
        }
    }
```

The meaning of these types can be defined as the followg:

SQLDataTypeDescriptor: This defines the data type of a single argument or result parameter.

colName: This specifies name of the result column.

varName: This specifies the name of the Data Variable as it appeared in the original SQL statement. This parameter must occur in SQL data type descriptors that occur in an SQLDBLArgumentSpecification. varName must not occur in SQL data type descriptors that occur within an SQLResultSpecification.

indName: This specifies the name of the Indicator Variable. If there was no Indicator Variable in the <variable specification>, then this parameter is omitted.

typeDescriptor: This defines the data type of the parameter in terms of database meta-data. There is a one-to-one correspondence between types in the typeDescriptor and items in SQL arguments and results. This correspondence is established through the ASN.1-type reference names: charType corresponds to characterItem, numericType corresponds to numericItem, etc. The type of the nth typeDescriptor must correspond with the type of the nth item in SQL arguments and results.

precision: For numericType and decimalType precision specifies the number of digits to a decimal base needed to represent the value transmitted. For integerType and smallIntType precision specifies the number of digits to the base as specified in precisionBase needed to represent the value transmitted.

scale: For numericType and decimalType scale specifies the power of 10 which is to be applied to compute the value.

precisionBase: For integerType and smallIntType precisionBase specifies the base used to count the number of digits when determining the precision which is represented.

The charSet parameter identifies the character set repertoire, and the accompanying length parameter specifies the maximum number of characters (not octets) allowed for the corresponding argument or result parameter.

If the charSet parameter in SQLDataTypeDescriptor is omitted, the character set used must be the default established by the declaration during the execution of of the R-Open service that opened the associated data resource. If no default was established, then the charSet parameter must be specified.

A charSet parameter is an Object Identifier uniquely identifying the specification of a coded character set which is described in the standard document ISO 8824:1989(E) (ASN.1).

If the descriptor entry specifies a variable length encoding, the value transmitted is equivalent to the same value as if the value had been transmitted is the same as the same value as if the value had been transmitted as a fixed length encoding and had been padded with SPACE's at the end of the string.

5.4.4 RDA SQL DBL Exceptions

```
SQLDBLException     ::= SEQUENCE
   {   sQLState             [0] IMPLICIT VisibleString OPTIONAL,
       sQLCode              [1] IMPLICIT INTEGER OPTIONAL,
       sQLErrorText         [2] IMPLICIT VisibleString OPTIONAL
       }
```

The meaning of these types can be defined as the following:

SQLDBLException: This parameter defines the completion code carried in the result for SQL DBL Operation. It is the method used by the SQL server to specify DBL exceptions.

sQLState: This value is defined in ISO/IEC DIS 9075:199x(E) or by the SQL server in conformance with that specification. sQLState must be returned by the SQL server if the applicable Level of Conformance specifies a year of 199x.

sQLCode: This value is defined by the SQL server in conformance with ISO/IEC 9075:1989(E). sQLCode must be returned by the SQL server if the applicable Level of Conformance specifies a year value of 1987 or 1989.

sQLErrorText: This value is a message describing the error or warning corresponding to the sQLState and the sQLCode values.

5.5 Protocol

This section describes the Server Execution rules required by the Specialization which are in addition to those defined in the ISO/IEC DIS 9579-1 (RDA Part 1: Generic).

5.5.1 Dialogue Management Services

Dialogue Initialization Functional Unit

R-Initialize Service

RDA Error Checking: If the value of the sQLConformanceLevel parameter does not specify a year value of 1987, 1989, or 199x, or if the year value is 199x and the level is not 1, then the error InvalidSQLConformanceLevel may be returned.

RDA Dialogue Entity Creation: The operation parameters values of sQLConformanceLevel and userdata must be copied to the attributes with the corresponding names in the newly-created RDA dialogue entity.

RDA Dialogue Entity manipulation: If one or more of the values for sQLConformanceLevel and userdata are not supported by the SQL server, the unsupported values must be changed to the values that are supported.

RDA Return: For each of the parameters in the SQLInitializeResult, if the corresponding value in the RDA Dialogue Entity was modified in the preceding step, its value from the RDA dialogue Entity must be included in the service result.

Dialogue Termination Functional Unit

R-Terminate service
No additional Server Execution Rules are required for the RDA SQL Specialization.

5.5.2 Transaction Management Services

Transaction Management Functional Unit

No additional Server Execution Rules for the all following services of this functional units are required.

R-BeginTransaction Service
R-Commit Service
R-Rollback Service

5.5.3 Control Services

Cancel Functional Unit

R-Cancel Service
If R-Cancel succeeds for a given operation X, then the effect on the underlying DBMS is as if the operation X were never executed.

R-status Service
No additional Server Execution Rules are required for the RDA SQL Specialization.

5.5.4 Resource Handling Services

Resource Handling Functional Unit

R-Open Service

RDA Error Checking

The SQLAccessControl value will be verified in conjunction with the dataResourceName supplied and the identityOfUser in the RDA Dialogue entity. If the SQLAccessControl value is not acceptable to the SQL server, the error SQLOpenError (sQLAccessControlViolation) must be returned.

If the SQLUsageMode value is not one of the legal values, then the error SQLOpenError (sQLUsageMode Violation) must be returned.

If an attempt is made to open another SQL Database resource before closing an already opened SQL Database Resources, the error SQLOpenError (sQLDatabaseResourceAlreadyOpenError) must be returned.

RDA Return

If the SQL Server cannot support the character set identified by the charSet argument in the service request, the SQL server must return the charSetNotSupported parameter.

R-Close Service

No additional Server Execution Rules are required for the RDA SQL Specialization.

5.5.5 Database Language Services

Immediate Execution DBL Functional Unit

R-ExecuteDBL service

An error will be returned if it is detected before the first execution of the DBL statement. Otherwise, the result will be returned. The result includes one or more SQLDBLException's depending on the repetitioCount or the list of SQLDBLArgumentValues.

RDA Error Checking

The SQLDBLArgumentSpecification, SQLDBLResultSpecification, SQLDBLstatement, and SQLDBLArgumentvalues parameters must be validated and any exceptions detected by the DBMS must be returned.

If the <host identifier> of an <embedded variable name> of the SQL statement is not "H", then the server returns the error hostIdentifierError.

If the SQL DB resource was opened in 'retrieval mode', and if the specified DBLstatement will update one or more objects in that SQL DB resource, then the error sQLUsageModeViolation must be returned.
If no execution of the statement completes, an error must be returned.
If an RDA client requests the execution of an SQL Data Statement and a transaction is not open at the RDA server, the RDA server must return the error rDATransactionNotOpen.

Execution
SQL statements are executed by the DBMS at the SQL server exactly as if they were embedded in a host program local to the DBMS.
An R-ExecuteDBL operation that executes an SQL Data Statement must be a part of a transaction.

RDA Result
For each execution of the DBLstatement, a specificDBLException returned by the DBMS must be included in the result. The specificDBLResult, if returned by the the DBMS, must also be included.

The following Table 5-2 shows the use of argument and result parameters for each SQL statement:

SQL Satement to be Executed	ArgSpec	ArgVal	ResSpec	ResVal
<close statement>				
<commit statement> 1				
<declare cursor>	C->S (H) 6			
<delete statement: positioned>				
<delete statement: searched>	C->S (H)	C->S (H)		
<fetch statement>			C->S 2	C<-S
<insert statement>	C->S (H)	C->S (H)		
<open statement>	C->S (H) 5	C->S (H)	C<-S 3	
<rollback statement>				
<select statement>	C->S (H)	C->S (H)	C<->S 4	C<-S
<update statement: positioned>	C->S (H)	C->S (H)		
<update statement: searched>	C->S (H)	C->S (H)		
<schema statement>				

Legend:
ArgSpec The SQLDBLArgumentSpecification parameter.
ArgVal The SQLDBLArgumentValues parameter.
ResSpec TheSQLDBLResultSpecification parameter.
ResVal The SQLDBLResultValues parameter.
C->S Parameter supplied by the client.
C<-S Parameter returned by the server.
(H) The client must set this parameter if the SQL statement contains host variables.
blank cells unspecified (I.e., not subject to conformance testing)

Comments:
1. If the content of SQLDBLstatement performs transaction management (e.g., SQL <commit statement> or SQL <rollback statement>), then the SQL server must return the error sQLDBLtransactionStatementNotAllowed. The client should use the Transaction Management services provided by RDA or TP depending on the Application Context chosen.
2. The client may optionally send the ResSpec, which will supercede the Resspec sent on a previous <fetch statement> using the same cursor or received from the server on the corresponding <open statement>.
3. The Resspec describes the ResVal that subsequent <fetch statement> using this cursor will return, but any ResSpec specified with a <fetch statement> overrides this ResSpec.
4. The client may optionally send the ResSpec, otherwise the server must return the ResSpec which describes the ResVal, if any.
5. The client may optionally send the ArgSpec. If it is not specified, the server must use the ArgSpec sent on the previous <declare cursor> statement that used the same cursor name.
6. The client may optionally send the Argspec. If it is not specified, the client must send the ArgSpec on the succeeding <open statement> that uses the same cursor name.

Table 5-2 Use of SQL Argument and Result Parameters in R-ExecuteDBL Service

Stored Execution DBL Functional Unit

R-DefineDBL Service

RDA Error Checking

The SQLDBLArgumentSpecification, SQLDBLResultSpecification, and statement parameters must be validated and any exceptions detected by the DBMS must be returned.
If the <host identifier> of an <embedded variable name> of the SQL statement is not "H", then the server returns the error hostIdentifierError.
If the SQL DB resource was opened in 'retrieval' mode, and if the specified DBLstatement will update one or more objects in that SQL DB resource then the errorsQLUsageModeViolation may be returned.

RDA Return
The following Table 5-3 shows the use of argument and result parameters for each SQL statement.

SQL Satement to be Executed	ArgSpec	ResSpec
<close statement>		
<commit statement> 1		
<declare cursor>	C->S (H)	
<delete statement: positioned>		
<delete statement: searched>	C->S (H)	
<fetch statement>		C->S 2
<insert statement>	C->S (H)	
<open statement>	C->S (H) 3	
<rollback statement>		
<select statement>	C->S (H)	C->S 2
<update statement: positioned>	C->S (H)	
<update statement: searched>	C->S (H)	
<schema statement>		

Legend:
ArgSpec The SQLDBLArgumentSpecification parameter.
ResSpec TheSQLDBLResultSpecification parameter.
C->S Parameter supplied by the client.
C<-S Parameter returned by the server.
(H) The client must set this parameter if the SQL statement contains host variables.
blank cells unspecified (I.e., not subject to conformance testing)

Comments:
1. If the content of SQLDBLStatement performs transaction management (e.g., SQL <commit statement> or <rollback statement>), the SQL server must return the error sQLDBLTransactionStatementNotAllowed. The client should use the Transaction Management services provided by RDA or TP depending on the Application Context chosen.
2. The client may optionally send the ResSpec to be used later when the operation is invoked using R-InvokeDBL.
3. The client may optionally send the ArgSpec. If it is not specified, the server must use the ArgSpec sent on the previous <declare cursor> statement that used the same cursor name.

Table 5-3 Use of SQL Argument and Result Parameters in R-DefineDBL Service

R-InvokeDBL Service

An error will be returned if it is detected before the first execution of the DBL operation identified by the commandHandle. Otherwise, the result will be returned. The result includes one or more SQLDBLException's depending on the repetitionCount or the list of SQLDBLArgumentValues.

RDA Error Checking

The SQLDBLArgumentValues parameter must be validated and any exception detected by the DBMS must be returned.

If the SQL DB resource was opened in 'retrieval' mode, and if the specified DBLstatement will update one or more objects in that SQL DB resource then the error sQLUsageModeViolation must be returned.

If an RDA client requests the execution of an SQL Data Statement and a transaction is not open at the RDA server, then the RDA server must return the error rDATransactionNotOpen.

Execution

An R-InvokeDBL operation that executes an SQL Data Statement must be a part of a transaction.

RDA Result

For each execution of the DBLstatement, a specificDBLException returned by the DBMS must be included in the result. The specificDBLResult, if returned by the DBMS, must also be included.

The following Table 5-4 shows the use of argument and result parameters for each SQL statement:

SQL Satement to be Executed	ArgVal	ResSpec	ResVal
<close statement>			
<commit statement> 1			
<declare cursor>			
<delete statement: positioned>			
<delete statement: searched>	C->S (H)		
<fetch statement>		1	C<-S
<insert statement>	C->S (H)		
<open statement>	C->S (H)	C<-S 2	
<rollback statement>			
<select statement>	C->S (H)	C<-S 3	C<-S
<update statement: positioned>	C->S (H)		
<update statement: searched>	C->S (H)		
<schema statement>			

Legend:
ArgVal The SQLDBLArgumentValues parameter.
ResSpec TheSQLDBLResultSpecification parameter.
ResVal The SQLDBLResultValues parameter.
C->S Parameter supplied by the client.
C<-S Parameter returned by the server.
(H) The client must set this parameter if the SQL statement contains host variables.
blank cells unspecified (I.e., not subject to conformance testing)

Comments:
1. The server does not return any ResSpec, but uses the one supplied by the client when the statement was defined using R-DefineDBL. If the client did not supply ResSpec on the R-DefineDBL request, the ResSpec returned on the invocation of the corresponding <open statement> is used.
2. The ResSpec describes the ResVal thaat subsequent invocation of a <fetch statement> using this cursor will return, but any ResSpec specified with the <fetch statement> on the R-DefineDBL request overrides this ResSpec.
3. If the client did not supply a ResSpec when the statement was defined, the server must return the ResSpec which describes the ResVal, if any.

Table 5-4 Use of SQL Argument and Result Parameters in R-InvokeDBL Service

R-DropDBL Service

The R-DropDBL operation has no effect on the state of the underlying DB. No additional Server Execution Rules are required for the RDA SQL Specialization.

5.5.6 Structure and Encoding of RDA SQL APDUs

The RDA SQL data types are described in this section by the ASN.1 notation specified in ISO 8824:1989(E) (ASN.1). The set of these RDA SQI APDUs defines the abstract syntax of the RDA SQL Presentation context. The transfer syntax for these RDA SQL APDUs is defined in the presentation context for the particular presentation connection.

Abstract Syntax Name

The ASN.1 object identifier value

{ iso standard rda (9579) part (2) abstract-syntax (1) version (1) }

as an abstract syntax name for the set of presentation data values, each of which is a value of the ASN.1 type ISO9579-RDASQL.RDA-APDU. The corresponding ASN.1 object descriptor value is

"RDA-SQL-ABSTRACT-SYNTAX-V1"

The ASN.1 object identifier and object descriptor values

{ joint-iso-ccitt asn (1) basic-encoding (1) }
and
"Basic Encoding of a single ASN.1 type"

(assigned to an information object in ISO 8825:1989(E) (ASN.1)) can be used as a transfer syntax name with this abstract syntax name.

ASN.1 Module for RDA SQL Specialization ASE

The Appendices show the ASN.1 Module for the RDA SQL Specialization ASE.

CHAPTER VI

RDA Service Interface and Server Modules Design

In this chapter, the design of RDA Service Interface and RDA server modules will be described. In the section 6.1, the design overview about about RDA Service Interface and RDA server is introduced. Then, the necessary modules of RDA Client Interface and RDA Server Interface for cooperating with their partners will be explained in section 6.2 and 6.3 respectively. A set of functions for RDA server which is working between the RDA Server Interface and a special DB server (i.e., SQL server) were designed to define more clearly the functionality of RDA server in section 6.4. Section 6.5 shows how these functions defined for RDA Service Interface and RDA server work in the overall view.

6.1 Overview

The RDA model (Figure 6-1) is composed of several components. Additionally, Figure 6-1 shows the SQL related components also for RDA specialization. The **RDA client** is the active RDA Communications Service user that initiates an RDA dialogue and requests DB access from a remote Database Server. The requests of RDA client pass through the **RDA client interface** which is the boundary between the RDA Communications Service and the RDA client. This is a part of the RDA Service Interface. The other passive RDA Communications Service user is the **RDA server** in a Database Server which provides DB access to remote RDA clients. The RDA server communicates with the RDA Communications Service via the **RDA Server Interface** which is a part of the RDA Service Interface and the boundary between the RDA Communications Service and the RDA server. The **RDA Protocol Machine** is the protocol machine of an RDA Application Service Element. The **SQL Data Resource** is an SQL DB as defined in ISO/IEC 9075 (SQL) which can be one of the possible data resources for the RDA. **SQL Server** is a conforming implementation of ISO/IEC 9075 (SQL) which provides the DB services at the RDA server.

The scenario of RDA model is as follows. The request of RDA client passes through the RDA client interface. Inside the RDA Communications Service, the request is transferred from RDA Client Protocol Machine to RDA Server Protocol Machine in the from of RI APDU. According to the indication primitive which is transferred to RDA server via RDA Server Interface, the RDA server generates the corresponding RDA operation. The actual operation execution is

requested to the SQL server via the SQL server interface to get the desired result by accessing the SQL data resource. The result of operation execution is returned to the RDA server by the SQL server via the SQL server interface. After getting the operation execution result, it is transferred to the RDA Server Protocol Machine via the RDA Server Interface as a response primitive. Then, the RDA Server Protocol Machine sends the operation execution result to the RDA Client Protocol Machine in the form of the RC APDU. Finally, the confirm primitive for the request of RDA client is returned to the RDA client by the RDA Client Protocol Machine via the RDA Client Interface.

The functionalities of two components of the RDA Service Interface, the RDA Client Interface and the RDA Server Interface will be defined and the corresponding library functions and the necessary parameters will be suggested in Section 6.2 and 6.3 respectively. Also, the functionality of RDA server and the corresponding library functions and the necessary parameters will be designed in Section 6.4. Based on those library functions, one working scenario of RDA will be explained finally.

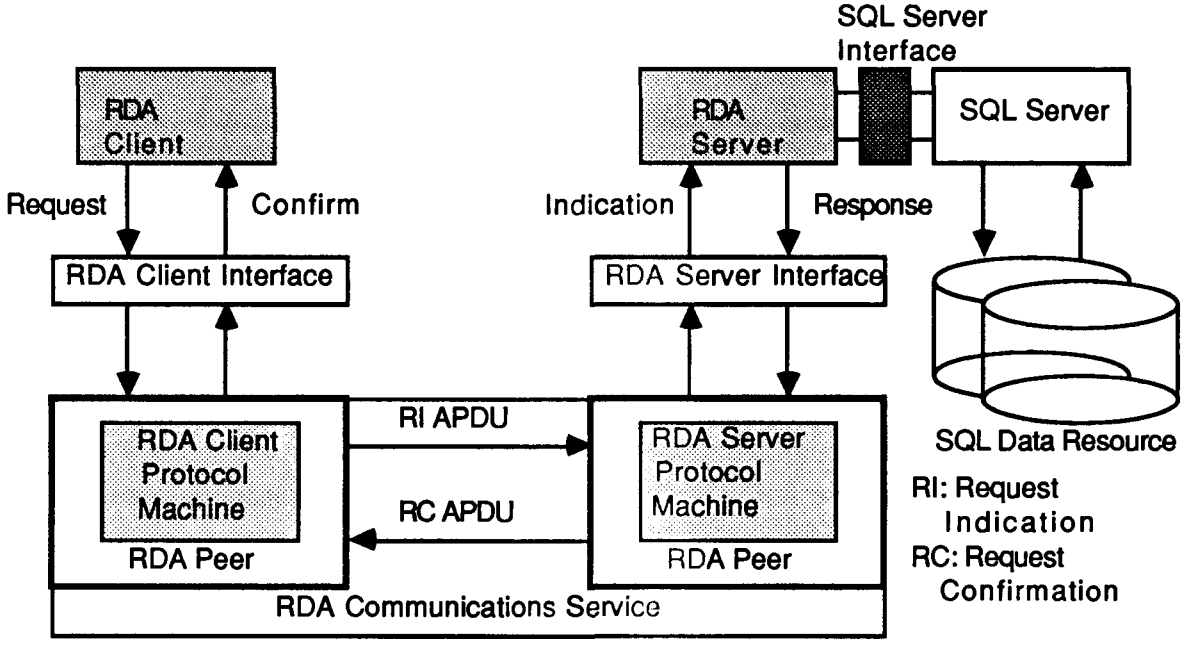


Figure 6-1 RDA Model

6.2 RDA Client Interface

The major functions of RDA Client Interface is receiving the request from the RDA client and sending the request to the RDA Client Protocol Machine and receiving the confirm for the request from the RDA Client Protocol Machine and sending the confirm to the client.

6.2.1 Design Model

The basic structure of RDA Client Interface (Figure 6-2) model has two important components, the RDA Client Interface endpoint (RDACIE) and RDA Client Interface queue (RDACIQ). The Client Interface has an endpoint is used as the access point for the RDA request and confirm primitives. The queue is used to store temporarily in scheduling the sending or receiving of the requests or confirms orderly. The functions for managing the events in the RDA Client Interface are stored in the function library. Each function is called to handle each step of RDA communication process.

The RDA Client Server Interface is operated in asynchronous mode. First, the request from RDA client is checked for the availability of the RDACIE. If the endpoint is available, it is enqueued into the RDA CIQ. If the queue is empty that time (no waiting requests or confirms) and the RDACIE is free then the dequed request from the RDACIQ is sent to the RDA Client Protocol Machine.

In case of RDA confirm, the internal mechanism is the same except the direction of data flow. The confirm from RDA Protocol Machine is checked for the availability of the RDACIE. If the endpoint is available, it is enqueued into the RDACIQ. If the queue is empty that time (no waiting requests or confirms) and the RDACIE is free then the confirm is sent to the RDA client.

When the RDACIQ is full or RDACIE is not free, the RDA client must wait until they become available. The event management function library for the RDA Client Interface has the functions for the checking of availability of those functions. If the RDACIE is free and the RDACIQ is not full then the RDA request/confirm is enqued until all the waiting events in the queue are processed and the RDACIE becomes available.

For the possible errors which can be occurred in the process of event processing, the error handling facility of event management function library will indicate the reason of errors and guide the directions to follow for the RDA client.

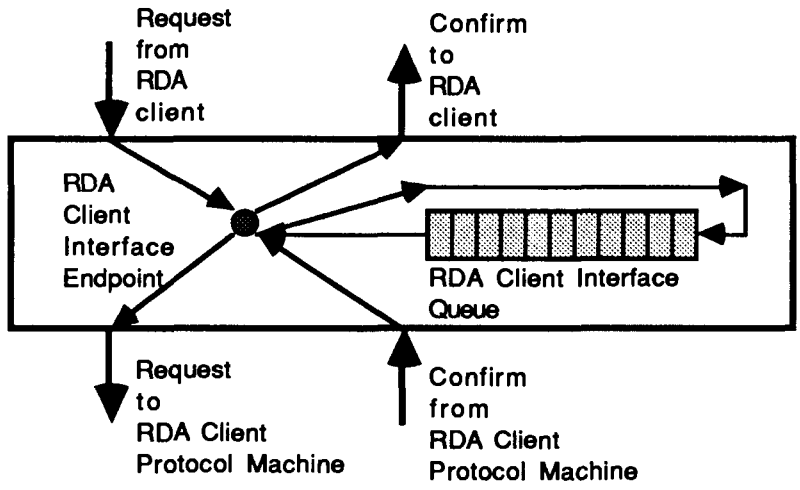


Figure 6-2 The Structure of RDA Client Interface

6.2.2 Library Functions and Parameters

The library functions for the event management of RDA Client Interface are as follows:

r_rcv_request() - receive the RDA request from the RDA client endpoint into the RDA Client Interface endpoint.

Input Parameters: r_c_endpoint - the endpoint of RDA client.
r_request - the RDA client request to the RDA server.
r_ci_endpoint - the endpoint of RDA Client Interface.

Return values: Upon success, puts the RDA client request to the endpoint of RDA Client Interface. On failure, error number is set to indicate an error.

Possible Errors: [RCIEPFULL] - The RDA Client Interface endpoint is not empty.
[RSYSERROR] - A system error has occurred during execution of function.

r_snd_request() - send the RDA request on the RDA Client Interface endpoint to the RDA client Protocol Machine endpoint.

Input Parameters: r_cpm_endpoint - the endpoint of the RDA Client Protocol Machine which is connected to the endpoint of the RDA Client Interface.
r_ci_endpoint - the endpoint of RDA Client Interface.

Return values: Upon success, sends the RDA client request on the RDACIE to the endpoint of RDA Client Protocol Machine.

Possible Errors: [RCNOEVENT] - There are no event on the RDACIE to be sent to the RDA Client Protocol Machine Endpoint (CPME).

[RCPMEPFULL] - The RDA CPME is occupied by the other event already.

[RSYSERROR] - A system error has occurred during execution of function.

r_rcv_confirm() - receive the RDA confirm from the RDA Client Protocol Machine endpoint into the RDA Client Interface endpoint.

Input Parameters: r_cpm_endpoint - the endpoint of the RDA Client Protocol Machine.

r_ci_endpoint - the endpoint of RDA Client Interface.

Return values: Upon success, receives the RDA confirm from the RDA Client Protocol Machine endpoint into the RDA Client Interface endpoint.

Possible Errors: [RCPMENOEVENT] - There are no event on the RDA CPM to be sent to the RDACIE.

[RSYSERROR] - A system error has occurred during execution of function.

r_snd_confirm() - send the RDA confirm from the RDA Client Interface endpoint to the RDA client endpoint .

Input Parameters: r_c_endpoint - the endpoint of RDA client.

r_ci_endpoint - the endpoint of RDA Client Interface.

Return values: Upon success, sends the RDA confirm from the RDA Client Interface endpoint into the RDA client endpoint.

Possible Errors: [RCIENOEVENT] - There are no event on the RDA Client Interface endpoint to be sent to the RDA client endpoint.

[RSYSERROR] - A system error has occurred during execution of function.

r_look_ci_endpoint() - returns the current event(s) associated with the RDA Client Interface endpoint.

Input Parameters: ci_endpoint - the endpoint of RDA Client Interface.

Return values: Upon success, sends the RDA returns a value that indicates which of the allowable events has occurred, or returns zero if no event exists.

Possible Errors: [RBADCIE] - The ci_endpoint does not refer to the endpoint of RDA Client Interface.

[RSYSERROR] - A system error has occurred during execution of function.

r_ci_transfer_event() - move an event from a source location to a destination location.

Input Parameters: source - the original location of event from which will be moved.
destination - the location which the event would be moved to.

Return values: Upon success, moves the event from the source location to the destination location.

Possible Errors: [RNOEVENT] - The source event to be moved does not exist.

[RINCOMPP] - The pointer types are not compatible.

[RSYSERROR] - A system error has occurred during execution of function.

r_ci_enqueue() - insert the event on the RDA Client Interface endpoint into the RDA Client Interface queue.

Input Parameters: r_ci_queue - the RDA Client Interface queue which is used to hold the events arrived.

r_ci_endpoint - the pointer which points the RDA Client Interface endpoint.

Return values: Upon success, inserts the event on the RDA Client Interface endpoint to the RDA Client Interface queue.

Possible Errors: [RCIQFULL] - The RDA Client Interface queue is full.

[RNOEVENT] - There are no event on the RDA Client Interface endpoint to be enqueued.

[RSYSERROR] - A system error has occurred during execution of function.

r_ci_dequeue() - get the event which is the first one from the RDA Client Interface queue.

Input Parameters: r_ci_queue - the RDA Client Interface queue which is used to hold the events arrived.

r_ci_endpoint - the pointer which points the RDA Client Interface endpoint.

Return values: Upon success, returns the event at the front of the the RDA Client Interface queue.

Possible Errors: [RCIQEMPTY] - The RDA Client Interface queue is empty.
[RNOEVENT] - There are no event on the RDA Client Interface endpoint to be enqueued.
[RSYSERROR] - A system error has occurred during execution of function.

r_open_ci_endpoint() - creates an RDACI endpoint and returns protocol-specific information associated with that point. It also returns a file descriptor that serves as the local identifier of the endpoint.

Input Parameters: r_name - points to an RDA Client Protocol Machine.
r_info - points to a structure r_info which contains the information about the local RDACI endpoint.

Return values: Upon success, a valid file descriptor that identifies a particular RDACI endpoint is returned.

Possible Errors: [RBADFLAG] - An invalid flag is specified.
[RBADCPMNAME] - Invalid Client Protocol Machine name.
[RSYSERROR] - A system error has occurred during execution of function.

r_close_ci_endpoint() - informs the RDA Client Protocol Machine that the RDA client is finished with the RDACI endpoint, and frees any local library resources associated with that endpoint. In addition, this function closes the file associated the the RDACI endpoint.

Input Parameters: fd - specifies a particular RDACI endpoint.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Client Interface endpoint.
[RSYSERROR] - A system error has occurred during execution of function.

r_bind_ci_endpoint() - associates a protocol address with a given RDACI endpoint, thereby activating the endpoint. It also directs the RDA Client Protocol Machine to begin accepting connect indications if desired.

Input Parameters: fd - specifies a particular RDACI endpoint.

req - is used to request that an address, represented by the predefined structure which contains the protocol address and connect indication information, be bound to the given RDACI endpoint.

ret - contains the address that the RDA Client Protocol Machine actually bound to the RDACI endpoint; this may be different from the address specified by the user in req.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Client Interface endpoint.

[RBADADDR] - The specified protocol address was in an incorrect format or contained illegal information.

[RADDRBUSY] - The address requested is in use and the RDA Client Protocol machine could not allocate a new address.

[RACCESS] - The client does not have permission to use the specified address.

[RSYSERROR] - A system error has occurred during execution of function.

r_unbind_ci_endpoint() - disables an RDACIE such that no further request destined for the given endpoint will be accepted by the RDA Client Protocol Machine

Input Parameters: fd - specifies a particular RDACI endpoint.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Client Interface endpoint.

[RWRONGSEQ] - The function was issued in the wrong sequence.

[RSYSERROR] - A system error has occurred during execution of function.

r_ci_optmgmt() - enables the client to get or negotiate protocol options with the RDA Client Protocol Machine

Input Parameters: fd - specifies a bound RDACI endpoint.
req - is used to request a specific action of the RDA Client Protocol Machine and to send options to the RDA Client Protocol Machine.
ret - contains the options and flag values to the RDA client.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Client Interface endpoint.
[RWRONGSEQ] - The function was issued in the wrong sequence.
[RBAOPT] - The specified protocol options were in an incorrect format or contained illegal information.
[RACCESS] - The client does not have permission to negotiate the specified options.
[RSYSERROR] - A system error has occurred during execution of function.

r_ci_getInfo() - returns protocol-specific service information associated with the specified RDA Client Interface endpoint.

Input Parameters: fd - specifies a bound RDACI endpoint.
r_info - used to return the same information returned by r_open.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Client Interface endpoint.
[RSYSERROR] - A system error has occurred during execution of function.

r_ci_sync() - synchronizes the data structures (the RDA Client Interface queue) managed by the RDA Client Interface library with the RDA Client Protocol Machine for the RDA client specified by the fd.

Input Parameters: fd - specifies an RDACI endpoint.

Return values: Upon success, the state of the RDA CI endpoint is returned. A value of -1 is returned, otherwise.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Client Interface endpoint.
[RTRANSITION] - The RDACI endpoint is under a state transition.
[RSYSERROR] - A system error has occurred during execution of function.

r_ci_alloc() - dynamically allocates storage for the various function argument data structure for the RDA Client Interface. This function will allocate memory for the specified structure, and will also allocate memory for buffers referenced by the structure.

Input Parameters: fd - specifies an RDACI endpoint.
r_struct_type - specifies the structure to allocate.
r_fields - specifies which buffers to allocate.

Return values: Upon success, r_ci_alloc returns a pointer to the newly allocated structure. On failure, a null pointer is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Client Interface endpoint.
[RBADSTRUCT] - Unsupported r_struct_type requested..
[RSYSERROR] - A system error has occurred during execution of function.

r_ci_free() - frees storage for a library data structure that was allocated by r_ci_alloc(). This function will free memory for the specified structure, and will also free memory for buffers referenced by the structure.

Input Parameters: r_ptr - points to the structure type described for r_alloc().
r_struct_type - specifies the type of a structure.

Return values: Upon success, a value of 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_ci_error() - prints out a message describing the last error encountered during a call to a Client User Interface library function.

Input Parameters: r_error_msg - is a user supplied error message that gives context to the error.

Return values: Upon success, a value of 0 is returned. Otherwise, alvalue of -1 is returned.

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

6.3 RDA Server Interface

The major functions of Server Interface is receiving the indication from the RDA Server Protocol Machine and sending the indication to the RDA server and receiving the response for the indication from the RDA server and sending the response to the RDA Server Protocol Machine.

6.3.1 Design Model

The basic structure of Server Interface (Figure 6-3) model has two important components, the RDA Server Interface endpoint (RDASIE) and RDA Server Interface queue (RDASIQ). The Client Interface has an endpoint is used as the access point for the RDA indication and response primitives. The queue is used to store temporarily in scheduling the sending or receiving of the indications or responses orderly. The functions for managing the events in the Server Interface are stored in the function library. Each function is called to handle each step of RDA communication process.

The RDA Server Interface is operated in asynchronous mode. First, the indication from RDA Server Protocol Machine is checked for the availability of the RDASIE. If the endpoint is available, it is enqueued into the RDASIQ. If the queue is empty that time (no waiting responses or indications) and the RDASIE is free then the dequed request from the RDASIQ is sent to the RDA Server endpoint.

In case of RDA response, the internal mechanism is the same except the direction of data flow. The response from RDA server is checked for the availability of the RDASIE. If the endpoint is available, it is enqueued into the RDASIQ. If the queue is empty that time (no waiting indications or responses) and the RDASIE is free then the response is sent to the RDA RDA Server Protocol Machine.

When the RDASIQ is full or RDASIE is not free, the RDA server must wait until they become available. The event management function library for the RDA Server Interface has the

functions for the checking of availability of those functions. If the RDASIE is free and the RDASIQ is not full then the RDA request/confirm is enqued until all the waiting events in the queue are processed and the RDASIE becomes available.

For the possible errors which can be occurred in the process of event processing, the error handling facility of event management function library will indicate the reason of errors and guide the directions to follow for the RDA client.

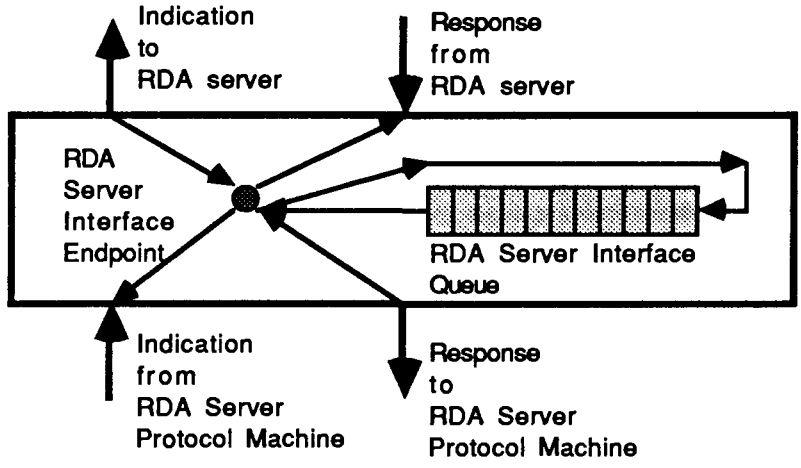


Figure 6-3 The Structure of RDA Server Interface

6.3.2 Library Functions and Parameters

The library functions for the event management of RDA Server Interface are as follows:

r_rcv_indication() - receive the RDA indication from the RDA Client Protocol Machine endpoint into the RDA Server Interface endpoint.

Input Parameters: r_spm_ endpoint - the endpoint of RDA Server Protocol Machine.
indication - the RDA indication to the RDA server.
si_endpoint - the endpoint of RDA Server Interface.

Return values: Upon success, puts the RDA indication to the endpoint of RDA Server Interface. On failure, error number is set to indicate an error.

Possible Errors: [RSIEPFULL] - The RDA Service Interface endpoint is not empty.

[RSYSERROR] - A system error has occurred during execution of function.

r_snd_indication() - send the RDA indication on the RDA Server Interface endpoint to the RDA server endpoint.

Input Parameters: r_spm_endpoint - the endpoint of the RDA Server Protocol Machine which is connected to the endpoint of the RDA Server Interface.
r_si_endpoint - the endpoint of RDA Server Interface.

Return values: Upon success, sends the RDA indication on the RDACIE to the endpoint of RDA Server.

Possible Errors: [RSNOEVENT] - There are no event on the RDASIE to be sent to the RDA server.
[RSEPFULL] - The RDA server endpoint is occupied by the other event already.
[RSYSERROR] - A system error has occurred during execution of function.

r_rcv_response() - receive the RDA response from the RDA server endpoint into the RDA Server Interface endpoint.

Input Parameters: r_s_endpoint - the endpoint of the RDA server.
r_si_endpoint - the endpoint of RDA Server Interface.

Return values: Upon success, receives the RDA response from the RDA server endpoint into the RDA Server Interface endpoint.

Possible Errors: [RCPMENOEVENT] - There are no event on the RDA server endpoint to be sent to the RDASIE.
[RSYSERROR] - A system error has occurred during execution of function.

r_snd_response() - send the RDA response on the RDA Server Interface endpoint to the RDA Server Protocol Machine endpoint.

Input Parameters: r_spm_endpoint - the endpoint of RDA Server Protocol Machine.
r_si_endpoint - the endpoint of RDA Server Interface.

Return values: Upon success, sends the RDA response from the RDA Server Interface endpoint into the RDA Server Protocol Machine endpoint.

Possible Errors: [RCIENOEVENT] - There are no event on the RDA Server Interface endpoint to be sent to the RDA Server Protocol Machine endpoint.
[RSYSERROR] - A system error has occurred during execution of function.

r_look_si_endpoint() - look at the current event on the RDA Server Interface endpoint.

Input Parameters: si_endpoint - the endpoint of RDA Server Interface.

Return values: Upon success, sends the RDA returns a value that indicates which of the allowable events has occurred, or returns zero if no event exists.

Possible Errors: [RBADSIE] - The si_endpoint does not refer to the endpoint of RDA Server Interface.
[RSYSERROR] - A system error has occurred during execution of function.

r_si_transfer_event() - move an event from a source location to a destination location.

Input Parameters: source - the original location of event from which will be moved.
destination - the location which the event would be moved to.

Return values: Upon success, moves the event from the source location to the destination location.

Possible Errors: [RNOEVENT] - The source event to be moved does not exist.
[RINCOMPP] - The pointer types are not compatible.
[RSYSERROR] - A system error has occurred during execution of function.

r_si_enqueue() - insert the event on the RDA Server Interface endpoint into the RDA Client Interface queue.

Input Parameters: r_si_queue - the RDA Server Interface queue which is used to hold the events arrived.
r_si_endpoint - the pointer which points the RDA Server Interface endpoint.

Return values: Upon success, inserts the event on the RDA Server Interface endpoint to the RDA Server Interface queue.

Possible Errors: [RSIQFULL] - The RDA Server Interface queue is full.

[RNOEVENT] - There are no event on the RDA Server Interface endpoint to be enqueued.

[RSYSERROR] - A system error has occurred during execution of function.

r_si_dequeue() - get the event which is the first one from the RDA Server Interface queue.

Input Parameters: r_si_queue - the RDA Server Interface queue which is used to hold the events arrived.

r_si_endpoint - the pointer which points the RDA Server Interface endpoint.

Return values: Upon success, returns the event at the front of the the RDA Server Interface queue.

Possible Errors: [RSIQEMPTY] - The RDA Server Interface queue is empty.

[RNOEVENT] - There are no event on the RDA Server Interface endpoint to be enqueued.

[RSYSERROR] - A system error has occurred during execution of function.

r_open_si_endpoint() -creates an RDASI endpoint and returns protocol-specific information associated with that point. It also returns a file descriptor that serves as the local identifier of the endpoint.

Input Parameters: r_name - points to an RDA Server Protocol Machine.

r_info - points to a structure r_info which contains the information about the local RDASI endpoint.

Return values: Upon success, a valid file descriptor that identifies a particular RDASI endpoint is returned.

Possible Errors: [RBADFLAG] - An invalid flag is specified.

[RBADSPMNAME] - Invalid Server Protocol Machine name.

[RSYSERROR] - A system error has occurred during execution of function.

r_close_si_endpoint() - informs the RDA Server Protocol Machine that the RDA server is finished with the RDASI endpoint, and frees any local library resources associated with that endpoint. In addition, this function closes the file associated the the RDASI endpoint.

Input Parameters: fd - specifies a particular RDASI endpoint.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Server Interface endpoint.

[RSYSERROR] - A system error has occurred during execution of function.

r_bind_si_endpoint() - associates a protocol address with a given RDASI endpoint, thereby activating the endpoint. It also directs the RDA Server Protocol Machine to begin accepting connect indications if desired.

Input Parameters: fd - specifies a particular RDASI endpoint.

req - is used to request that an address, represented by the predefined structure which contains the protocol address and connect indication information, be bound to the given RDASI endpoint.

ret - contains the address that the RDA Server Protocol Machine actually bound to the RDASI endpoint; this may be different from the address specified by the user in req.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Server Interface endpoint.

[RBADADDR] - The specified protocol address was in an incorrect format or contained illegal information.

[RADDRBUSY] - The address requested is in use and the RDA Server Protocol machine could not allocate a new address.

[RACCESS] - Th client does not have permission to use the specified address.

[RSYSERROR] - A system error has occurred during execution of function.

r_unbind_si_endpoint() - disables an RDASIE such that no further request destined for the given endpoint will be accepted by the RDA Server Protocol Machine

Input Parameters: fd - specifies a particular RDASI endpoint.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Server Interface endpoint.

[RWRONGSEQ] - The function was issued in the wrong sequence.

[RSYSERROR] - A system error has occurred during execution of function.

r_si_optmgmt() - enables the client to get or negotiate protocol options with the RDA Server Protocol Machine

Input Parameters: fd - specifies a bound RDASI endpoint.

req - is used to request a specific action of the RDA Server Protocol Machine and to send options to the RDA Server Protocol Machine.

ret - contains the options and flag values to the RDA server.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Server Interface endpoint.

[RWRONGSEQ] - The function was issued in the wrong sequence.

[RBAOPT] - The specified protocol options were in an incorrect format or contained illegal information.

[RACCESS] - The client does not have permission to negotiate the specified options.

[RSYSERROR] - A system error has occurred during execution of function.

r_si_getinfo() - returns protocol-specific service information associated with the specified RDA Server Interface endpoint.

Input Parameters: fd - specifies a bound RDASI endpoint.

r_info - used to return the same information returned by r_open.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Server Interface endpoint.

[RSYSERROR] - A system error has occurred during execution of function.

r_si_sync() - synchronizes the data structures (the RDA Server Interface queue) managed by the RDA Server Interface library with the RDA Server Protocol Machine for the RDA server specified by the fd.

Input Parameters: fd - specifies an RDASI endpoint.

Return values: Upon success, the state of the RDASI endpoint is returned. A value of -1 is returned, otherwise.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Server Interface endpoint.
[RTRANSITION] - The RDASI endpoint is under a state transition.
[RSYSERROR] - A system error has occurred during execution of function.

r_si_alloc() - dynamically allocates storage for the various function argument data structure for the RDA Server Interface. This function will allocate memory for the specified structure, and will also allocate memory for buffers referenced by the structure.

Input Parameters: fd - specifies an RDASI endpoint.
r_struct_type - specifies the structure to allocate.
r_fields - specifies which buffers to allocate.

Return values: Upon success, r_ci_alloc returns a pointer to the newly allocated structure. On failure, a null pointer is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an RDA Server Interface endpoint.
[RBADSTRUCT] - Unsupported r_struct_type requested..
[RSYSERROR] - A system error has occurred during execution of function.

r_si_free() - frees storage for a library data structure that was allocated by r_si_alloc(). This function will free memory for the specified structure, and will also free memory for buffers referenced by the structure.

Input Parameters: r_ptr - points to the structure type described for r_si_alloc().
r_struct_type - specifies the type of a structure.

Return values: Upon success, a value of 0 is returned. Otherwise, alvalue of -1 is returned.

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_si_error() - prints out a message describing the last error encountered during a call to a Server Interface library function.

Input Parameters: r_error_msg - is a user supplied error message that gives context to the error.

Return values: Upon success, a value of 0 is returned. Otherwise, alvalue of -1 is returned.

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

6.4 RDA Server Modules

The RDA server can be composed of a set of functions. The first group of functions are related with the creation, deletion and manipulation of RDA entities and attributes of RDA Dialogue State model. The second group of functions are related with the interface for database (i.e., SQL) server. The third group of functions are concerned with the interface with the RDA Server Interface.

Each function group is closely connected with each other. The actual database access can be done at the data resource of database server. From the RDA server, the corresponding database operation is asked to be executed by the database server according to the type of RDA operation. The database execution result of database server (i.e., SQL Server) is returned via the database server interface (i.e., SQL Server Interface) to the RDA server. The RDA server transfers this result to the RDA Server Interface endpoint via the RDA Server endpoint. Finally, this response for the RDA client's request is sent to the TRDA client via the RDA Communications Service.

6.4.1 Design Model

An RDA client's request is responded by an RDA server. These requests are communicated to the RDA server as indication events at the RDA Server Interface. These events, together with the result and error responses, comprise the RDA Server Interface.

Inside the server, the requests received form the client are modelled as **operation entities** that go through the server until they return as result or error response primitive events at the RDA Server Interface.

The semantics of operation entities and their processing by the database server are defined by a model of the state of server and by the rules governing the progression of operation entities through the server.

The state of the server has two main components:

* The dialogue state - models the interactions with a single client over one dialogue.
* The database state - models persistent (stored) data which is shared among all the DB users. This state is changed by DB operations that are defined by RDA specializations.

The server execution rules describe the permitted effects of execution of operations by the RDA server.

* Describe the changes to dialogue state resulting from execution of an operation entity.
* Define the conditions that apply when a result is generated.
* Define the conditions that apply when an error is generated.

The main focus of design model for the RDA server is the RDA operations for the RDA entity and attribute manipulations of Dialogue State Model. The general state transition of operation in the RDA server is depicted in Figure 6-4.

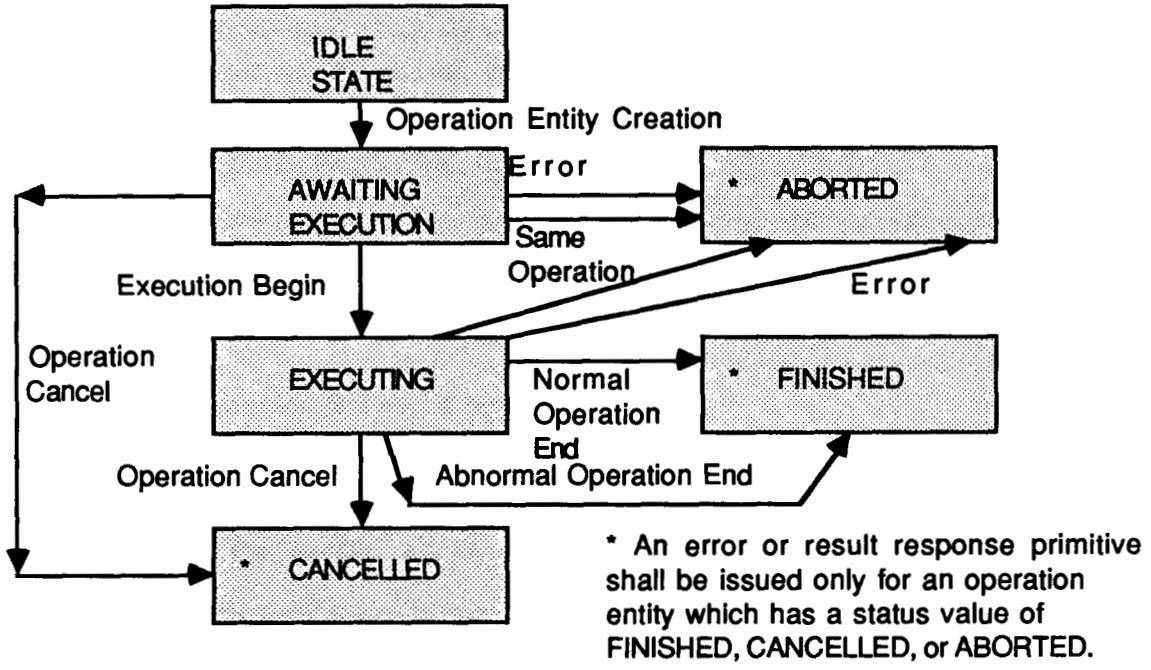


Figure 6-4 State Transition of Operation in the RDA Server

In related with Figure 6-4, the followings are the general server execution rules which apply to all service requests by an RDA server. The general rules provide for specific types of augmentation based on the particular service request type.

* Operation Entity Creation

When an RDA server receives a service indication primitive it creates an operation entity with the following constraints on the initial attribute values:

Attribute	Initial Value
dialogueID	The dialogueID for the current dialogue.
operationID	The operationID parameter on the service indication primitive.
status	AWAITING EXECUTION
cancelRequestReceived	FALSE
operationInvocationType	The type of service invocation primitive.
operationArgument	The parameters on the service indication primitive.
operationResult	NULL
operationError	NULL

For the operation entity created, if there exists another operation entity with the same value for the dialogueID attribute and the same value for the operationID attribute, then newly created operation entity will be modified as the following if the value of it status attribute is AWAITING EXECUTION or EXECUTING:

Attribute	New Value
status	ABORTED
operationError	DuplicateOperationID

* Implementor Defined Errors

If any implementor defined error occurs during AWAITING EXECUTION or EXCUTING, the operation entity which has the AWAITING EXECUTION or EXECUTING status value and whose operationInvocationType permits an error response of OperationAborted will be modified as following:

Attribute	New Value
status	ABORTED
operationError	OperationAborted with supplemental information

* Operation Begin

When an operation with AWAITING EXECUTION status value begins executing, its operation entity is modified as follows:

Attribute	New Value
status	EXECUTING

* Operation Cancel

After the RDA cancelled an executing operation to which R-Cancel service indication was issued, any operation entity having a status value of EXECUTING or AWAITING EXECUTION and a cancelRequestReceived attribute of TRUE will be modified as:

Attribute	New Value
status	CANCELLED
operationError	OperationCancelled

* Operation Execution

Any operation entity having the EXECUTING status attribute value will be modified according to the server execution rules for the particular operationInvocationType. The server

execution rules for a particular service include 3 general types of rules, i.e., Entity Modification Rules, Result Rules, Error Rules.

* Operation End

When an operation end abnormally, an operation entity having the status attribute value EXECUTING will be modified as the following:

Attribute	New Value
status	FINISHED
operationError	Any error response value which is allowed for the request type indicated by the operationInvocationType attribute and which is not prohibited by any prerequisite error predicate for the request type.

When an operation end normally, an operation entity having the status attribute value EXECUTING will be modified as the following:

Attribute	New Value
status	EXECUTING
operationResult	Any result response value which satisfies the constraints for the request type indicated by the operationInvocationType attribute provided no mandatory error predicate for the request type is satisfied.

* Response to an Operation

A result or error response primitive will be issued only for an operation entity which has a status value of FINISHED, CANCELLED, or ABORTED. The operation entity will be deleted when the response primitive is issued. The server can execute more than one RDA operation at the same time and the serializability of RDA operations is maintained except R-Cancel and R-Status operations.

* Association Failure

If the underlying association in use by an RDA dialogue fails, then all entities (dialogue, opened data resource, defined DBL, and operation) whose dialogueID attribute equals the dialogueID for the current dialogue will be deleted.

In Section 6.4.2, the interface for the SQL server will be explained. The generic object management library functions for the RDA server will be described in Section 6.4.3 The operation execution for each service request will be described in Section 6.4.4.

6.4.2 Interface to the SQL Server

For each function call from the server, there is an interface to the SQL server. This interface has the endpoint which is used as a point for the data exchange between the RDA server and the SQL server. All the operation requested by the RDA server will be executed by the SQL server and returned to the RDA server via the the SQL Server Interface.

6.4.3 Generic Object Management Library Functions and Parameters

The generic object management library functions for the RDA server are as follows:

The following functions are related with the creation, deletion and manipulation of RDA entities and attributes of RDA Dialogue State Model.

r_operation_entity_create() - create an operation entity with the constraints on the initial attribute values.

Input Parameters: r_indication_p - the service indication primitive.

Return values: Upon success, creates an operation entity with the following attribute values.

Attribute	Initial Value
dialogueID	The dialogueID for the current dialogue.
operationID	The operationID parameter on the service indication primitive.
status	AWAITING EXECUTION
cancelRequestReceived	FALSE
operationInvocationType	The type of service invocation primitive.
operationArgument	The parameters on the service indication primitive.
operationResult	NULL
operationError	NULL

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_operation_entity_delete() - delete an operation entity with attribute values.

Input Parameters: r_op_entity_ptr - the pointer to the operation entity to be deleted.

Return values: Upon success, deletes an operation entity which is pointed by r_op_entity_ptr.

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_dialogue_entity_create() -create an RDA dialogue entity with the constraints on the attribute values.

Input Parameters: r_indication_p - the service indication primitive which is for the R-Initialize service request.

Return values: Upon success, creates a dialogue entity with the following attribute values.
dialogueID, identityOfUser, controlServicesAllowed, controlAuthenticationData, functionalUnitsNegotiated, RDATransactionStatus, RDATransactionRolledBack

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_dialogue_entity_delete() - delete an RDA dialogue entity with attribute values.

Input Parameters: r_dialog_entity_ptr - the pointer to the dialog entity to be deleted.

Return values: Upon success, deletes a dialog entity which is pointed by r_dialog_entity_ptr.

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_ddbl_entity_create() - create a defined DBL entity with the constraints on the attribute values.

Input Parameters: r_indication_p - the service indication primitive which is for the R-DefineDBL service request.

Return values: Upon success, creates a defined DBL entity with the following attribute values.
dialogueID, commandHandle, dataResourceHandle

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_ddbl_entity_delete() - delete a defined DBL entity with attribute values.

Input Parameters: r_ddbl_entity_ptr - the pointer to the defined DBL entity to be deleted by the R-DropDBL service request.

Return values: Upon success, deletes a defined DBL entity which is pointed by r_ddbl_entity_ptr.

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_opened_dr_entity_create() - create an opened data resource entity with the constraints on the attribute values.

Input Parameters: r_indication_p - the service indication primitive which is for the R-Open service request.

Return values: Upon success, creates an opened data resource entity with the following attribute values.
dialogueID, dataResourceHandle, parentDataResourceHandle, dataResourceName

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_opened_dr_entity_delete() - delete an opened data resource entity with attribute values.

Input Parameters: r_opened_dr_entity_ptr - the pointer to the opened data resource entity to be deleted by the R-Close service request.

Return values: Upon success, deletes an opened data resource entity which is pointed by r_opened_dr_entity_ptr.

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

r_entity_modify() - modify a specific attribute value of RDA entity according to the given argument value.

Input Parameters: r_entity - the entity to be modified.
r_attr_list - the list of attributes to be modified.
r_attr_value_list - the attribute values for the r_attr_list.

Return values: Upon success, modifies the attribute values of attributes r_attr_list according to the new attribute values r_attr_value_list.

Possible Errors: [RTYPEMISMATCH] - An attribute value type is not compatible with the attribute type.
[RNOMISMATCH] - The number of attributes and attribute values are not matched.
[RSYSERROR] - A system error has occurred during execution of function.

r_implementor_error_handle() - prints out a message describing the last error encountered during a call to Generic Object Management Library function.

Input Parameters: r_error_msg - is an implementor supplied error message that gives context to the error.

Return values: Upon success, a value of 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RSYSERROR] - A system error has occurred during execution of function.

The following functions are related with the interface for database (i.e., SQL) server. The basic idea of this function set is sending the indication primitive for the client's request via the SQL Server Interface and receiving the execution result form the SQL server via SQL Server Interface. The role of each function is described briefly in the following except last few functions. For each service element of RDA, a pair of functions are necessary for sending/receiving the data.

r_initialize_send_sql_server() - send an R-Initialize operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_initialize_receive_sql_server() - receive an R-Iinitialize operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_terminate_send_sql_server() - send an R-Terminate operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_terminate_receive_sql_server() - receive an R-Terminate operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_begintransaction_send_sql_server() - send an R-BeginTransaction operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_begintransaction_receive_sql_server() - receive an R-BeginTransaction operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_commit_send_sql_server() - send an R-Commit operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_commit_receive_sql_server() - receive an R-Commit operation result from the SQL Server Interface endpoint into the RDA server

r_rollback_send_sql_server() - send an R-Rollback operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_rollback_receive_sql_server() - receive an R-Rollback operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_cancel_send_sql_server() - send an R-Cancel operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_cancel_receive_sql_server() - receive an R-Cancel operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_status_send_sql_server() - send an R-Status operation result on the RDA server endpoint to the SQL Server Interface endpoint

r_status_receive_sql_server() - receive an R-Status operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_open_send_sql_server() - send an R-Open operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_open_receive_sql_server() - receive an R-Open operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_close_send_sql_server() - send an R-Close operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_close_receive_sql_server() - receive an R-Close operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_executedbl_send_sql_server() - send an R-ExecuteDBL operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_executedbl_sql_server() - receive an R-ExecuteDBL operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_definedbl_send_sql_server() - send an R-DefineDBL operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_definedbl_receive_sql_server() - receive an R-DefineDBL operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_invokedbl_send_sql_server() - send an R-InvokeDBL operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_invokedbl_receive_sql_server() - receive an R-InvokeDBL operation result from the SQL Server Interface endpoint into the RDA server endpoint

r_dropdbl_send_sql_server() - send an R-DropDBL operation request on the RDA server endpoint to the SQL Server Interface endpoint

r_dropdbl_receive_sql_server() - receive an R-DropDBL operation result from the SQL Server Interface endpoint into the RDA server endpoint

The following functions are concerned with the interface with the SQL Server Interface.

r_open_sql_sever_endpoint() - creates an SQL Server Interface endpoint and returns protocol-specific information associated with that point. It also returns a file descriptor that serves as the local identifier of the endpoint.

Input Parameters: r_name - points to an RDA server.
r_info - points to a structure r_info which contains the information about the local SQL Server Interface endpoint.

Return values: Upon success, a valid file descriptor that identifies a particular SQL Server Interface endpoint is returned.

Possible Errors: [RBADFLAG] - An invalid flag is specified.
[RBADSERVERNAME] - Invalid RDA server name.
[RSYSERROR] - A system error has occurred during execution of function.

r_close_sql_server_endpoint() - informs the RDA server that the RDA server finished with the SQL server endpoint, and frees any local library resources associated with that endpoint. In addition, this function closes the file associated the the SQL Server Interface endpoint.

Input Parameters: fd - specifies a particular SQL server endpoint.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an SQL Server Interface endpoint.
[RSYSERROR] - A system error has occurred during execution of function.

r_bind_sql_server_endpoint ()-associates a protocol address with a given SQL server endpoint, thereby activating the endpoint. It also directs the RDA Server to begin accepting connect indications if desired.

Input Parameters: fd - specifies a particular SQL server endpoint.

req - is used to request that an address, represented by the predefined structure which contains the protocol address and connect indication information, be bound to the given SQL Server Interface endpoint.

ret - contains the address that the RDA Server actually bound to the SQL Server Interface endpoint; this may be different from the address specified by the user in req.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an SQL Server Interface endpoint.

[RBADADDR] - The specified protocol address was in an incorrect format or contained illegal information.

[RADDRBUSY] - The address requested is in use and the RDA Server could not allocate a new address.

[RACCESS] - The RDA server does not have permission to use the specified address.

[RSYSERROR] - A system error has occurred during execution of function.

r_unbind_sql_server_endpoint() - disables an SQL Server Interface endpoint such that no furtherrequest destined for the given endpoint will be accepted by the SQL server.

Input Parameters: fd - specifies a particular SQL Server Interface endpoint.

Return values: Upon success, a value 0 is returned. Otherwise, a value of -1 is returned.

Possible Errors: [RBADF] - The specified file descriptor does not refer to an SQL Server Interface endpoint.

[RWRONGSEQ] - The function was issued in the wrong sequence.

[RSYSERROR] - A system error has occurred during execution of function.

The following functions are concerned with the interface with the RDA Server Interface.

r_rcv_indication() - receive the RDA indication from the RDA Server Interface endpoint into the RDA Server endpoint

Input Parameters: r_si_endpoint - the endpoint of RDA Server Interface.
indication - the RDA indication to the RDA server.
r_server_endpoint - the endpoint of RDA server.

Return values: Upon success, puts the RDA indication to the endpoint of RDA Server endpoint. On failure, error number is set to indicate an error.

Possible Errors: [RSERVEREPFULL] - The RDA server endpoint is not empty.
[RSYSERROR] - A system error has occurred during execution of function.

r_snd_response() - send the RDA response on the RDA Server endpoint to the RDA Server Interface endpoint

Input Parameters: r_server_endpoint - the endpoint of RDA server.
r_si_endpoint - the endpoint of RDA Server Interface.

Return values: Upon success, sends the RDA response from the RDA server endpoint into the RDA Server Interface endpoint.

Possible Errors: [RSERVERENOEVENT] - There are no event on the RDA server endpoint to be sent to the RDA Server Interface endpoint.
[RSYSERROR] - A system error has occurred during execution of function.

6.4.4 Server Operation Executions for Each Service

Before the operation execution for service request, the RDA server calls the function **r_rcv_indication()** to receive the RDA indication from the RDA Server Interface endpoint into the RDA Server endpoint. The input parameters are the r_si_endpoint which is the endpoint of RDA Server Interface and indication which is the RDA indication to the RDA server and r_server_endpoint which is the endpoint of RDA server. This function puts the RDA indication to the endpoint of RDA Server endpoint in case of success. On failure, error number is set to indicate an error. After this function call, the RDA server calls a series of functions

form the Generic Object Management Library for each service request according to the RDA server rules.

After the function r_rcv_indication() call, the service indication is matched with the proper Generic Object Management Library function to perform the service requested by the RDA client. The following explains the operation of each RDA service according to the Generic Object Management Library functions of model developed for RDA server.

* R-Initialize Service Request

The R-Initialize service operation execution is depicted in Figure 6-5. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_dialogue_entity_create()
r_dialogue_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

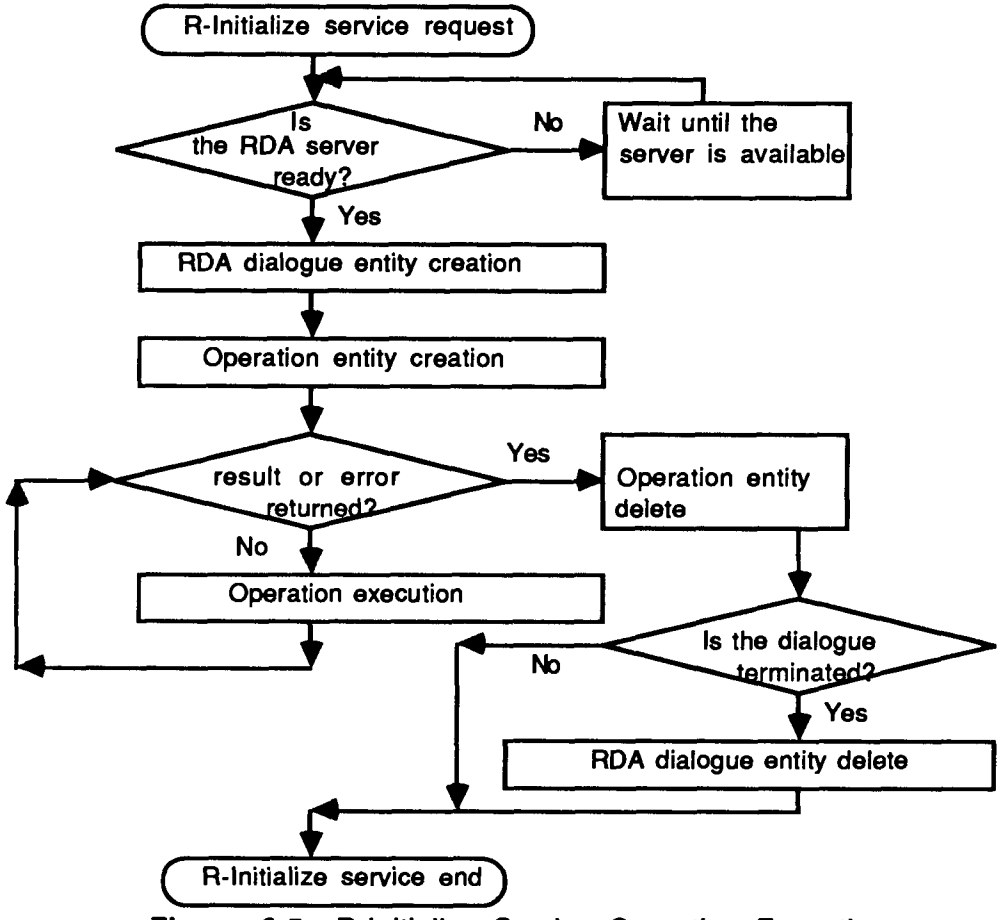


Figure 6-5 R-Initialize Service Operation Execution

* R-Terminate Service Request

The R-Terminate service operation execution is depicted in Figure 6-6. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_dialogue_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

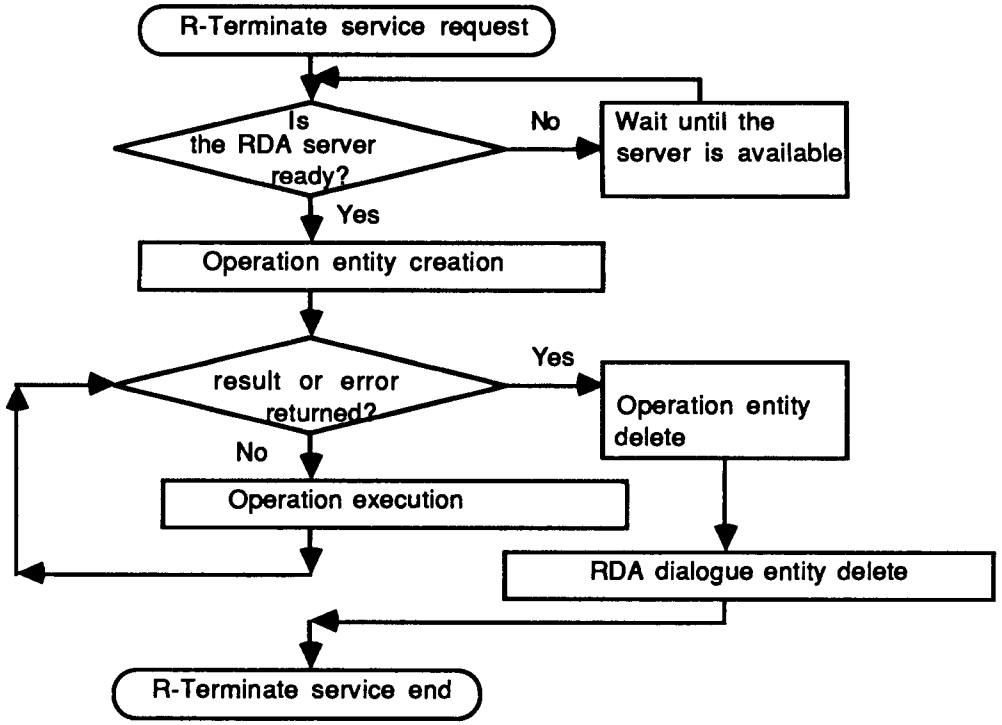


Figure 6-6 R-Terminate Service Operation Execution

* R-BeginTransaction Service Request

The R-BeginTransaction service operation execution is depicted in Figure 6-7. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

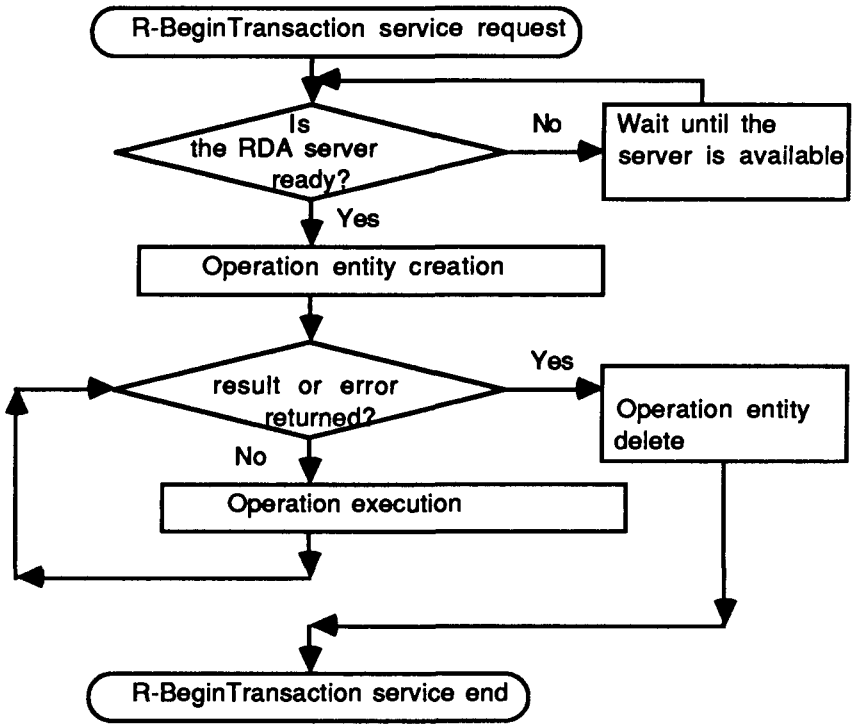


Figure 6-7 R-BeginTransaction Service Operation Execution

* R-Commit Service Request

The R-Commit service operation execution is depicted in Figure 6-8. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

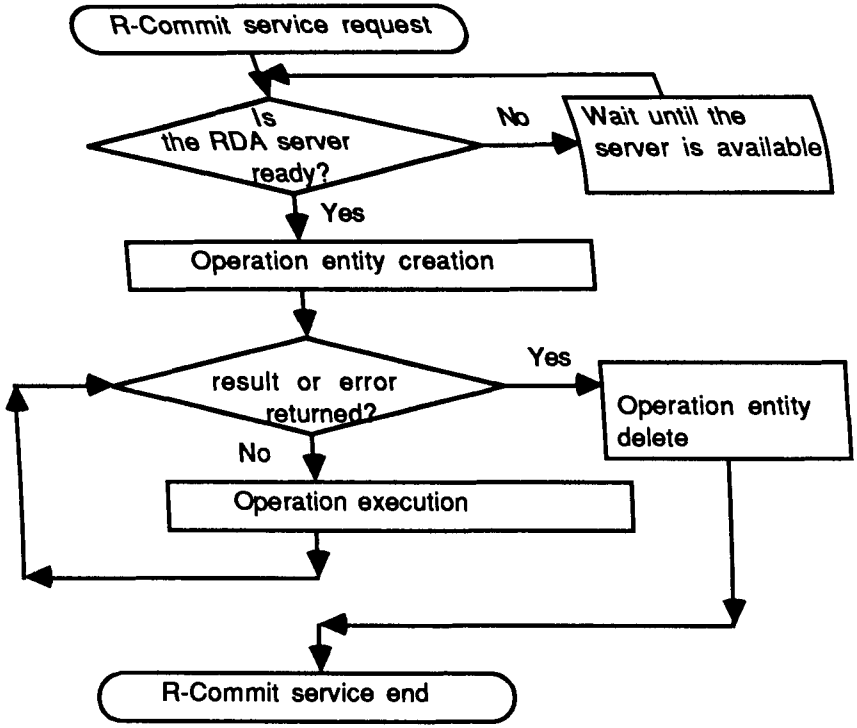


Figure 6-8 R-Commit Service Operation Execution

* R-Rollback Service Request

The R-Rollback service operation execution is depicted in Figure 6-9. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

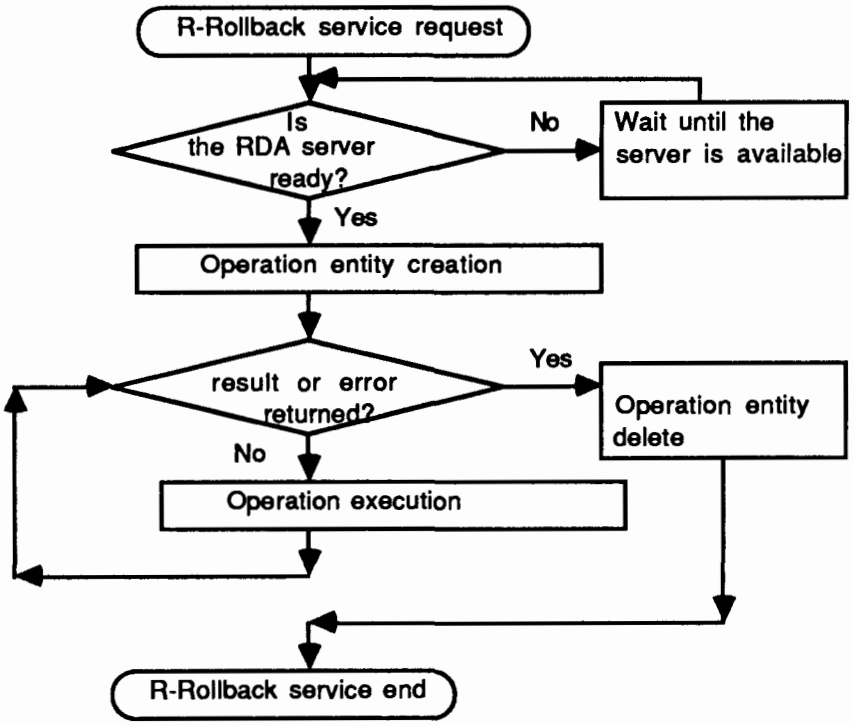


Figure 6-9 R-Rollback Service Operation Execution

* R-Cancel Service Request

The R-Cancel service operation execution is depicted in Figure 6-10. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

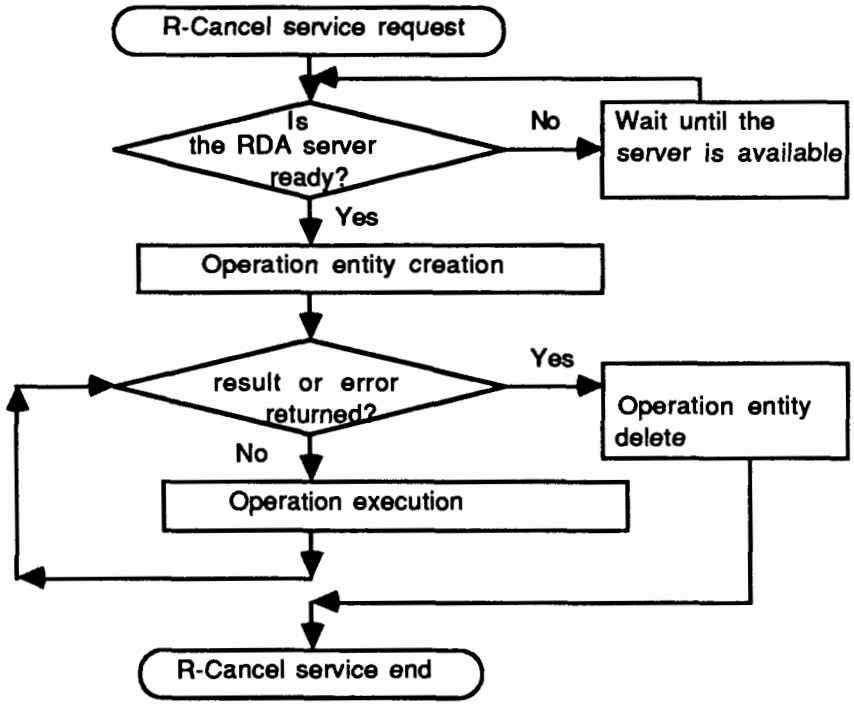


Figure 6-10 R-Cancel Service Operation Execution

* R-Status Service Request

The R-Status service operation execution is depicted in Figure 6-11. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

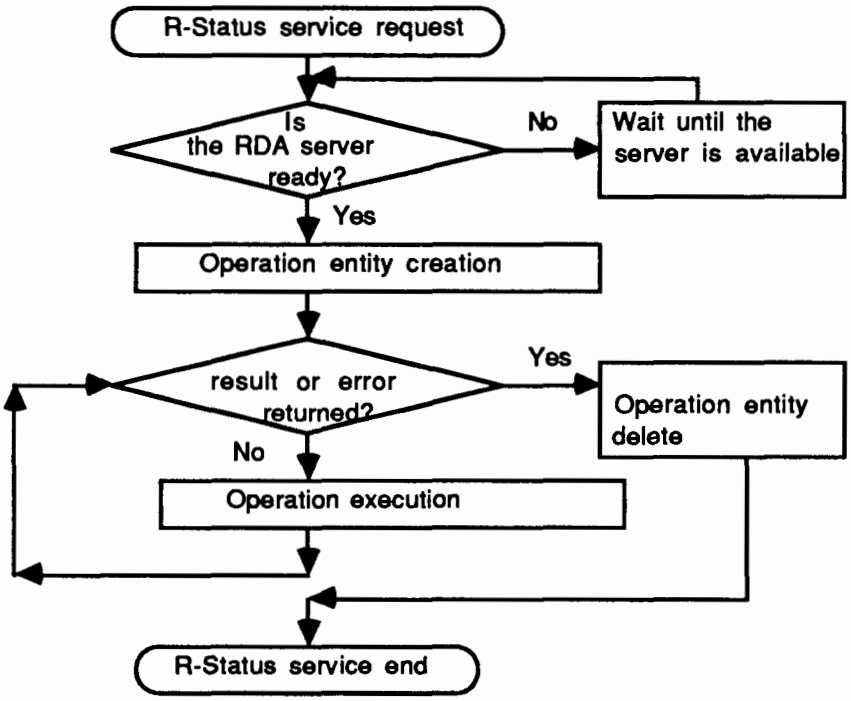


Figure 6-11 R-Status Service Operation Execution

* R-Open Service Request

The R-Open service operation execution is depicted in Figure 6-12. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_opened_dr_entity_create()
r_entity_moidify()
r_implementor_error_handle()

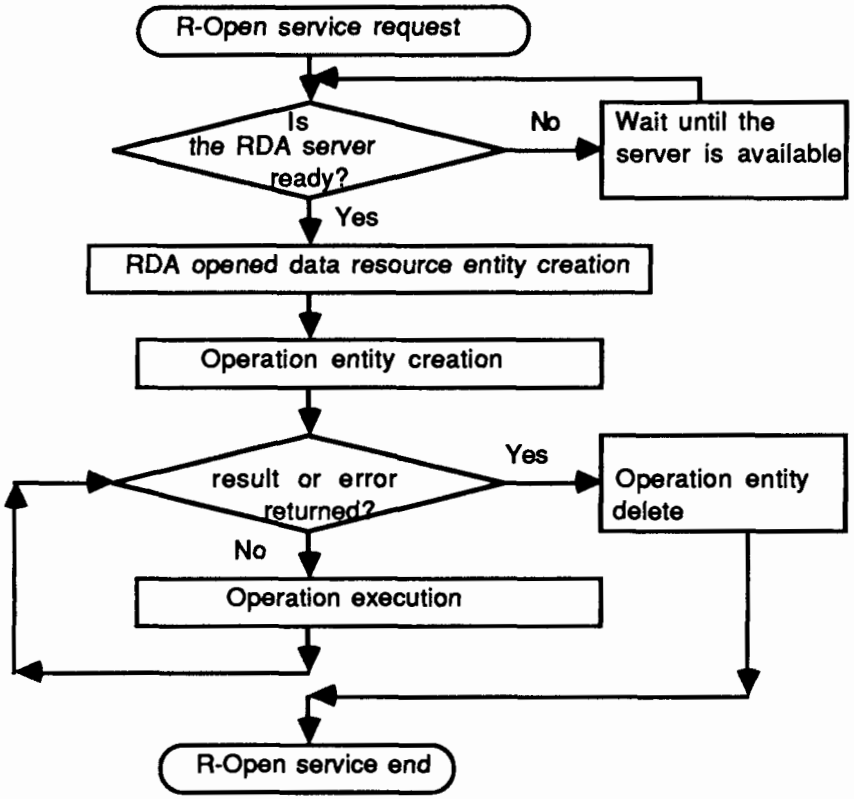


Figure 6-12 R-Open Service Operation Execution

* R-Close Service Request

The R-Close service operation execution is depicted in Figure 6-13. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_opened_dr_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

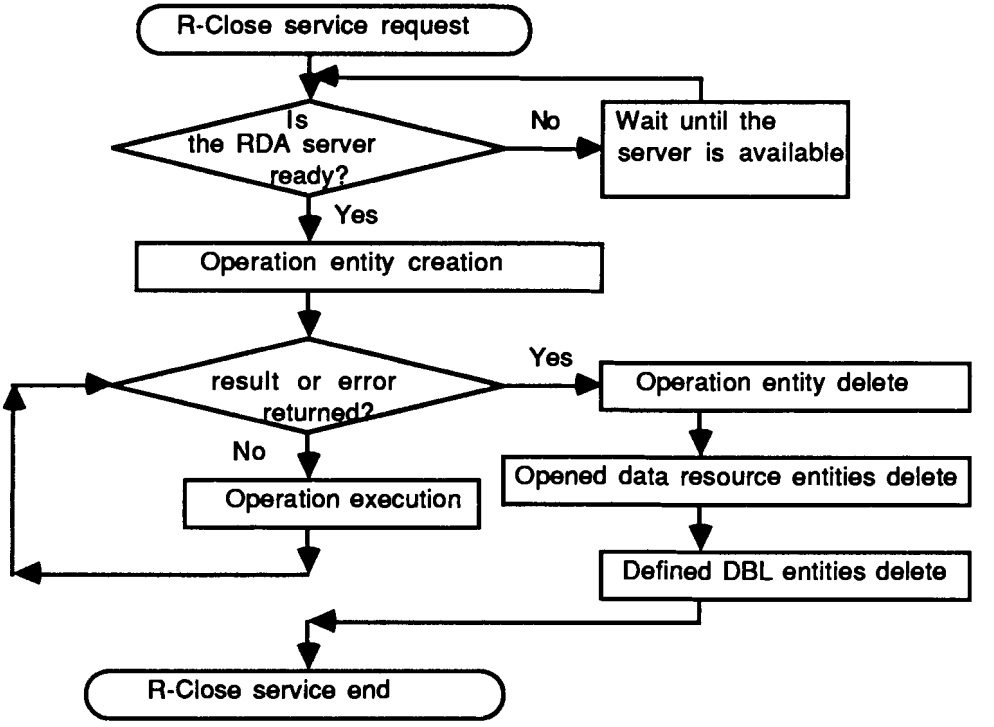


Figure 6-13 R-Close Service Operation Execution

* R-ExecuteDBL Service Request

The R-ExecuteDBL service operation execution is depicted in Figure 6-14. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

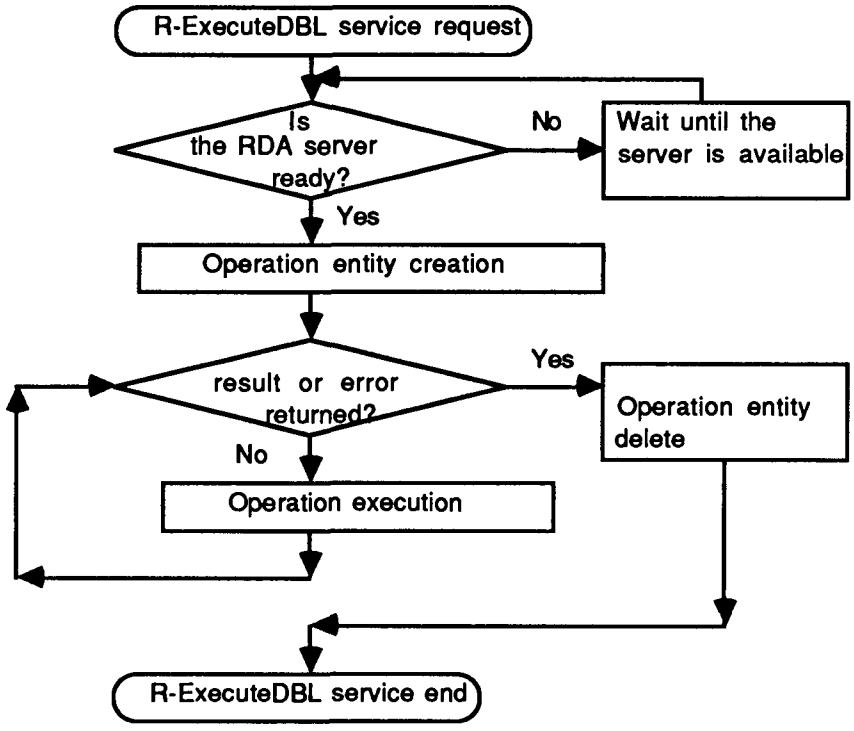


Figure 6-14 R-ExecuteDBL Service Operation Execution

* R-DefineDBL Service Request

The R-DefineDBL service operation execution is depicted in Figure 6-15. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_ddbl_entity_create()
r_ddbl_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

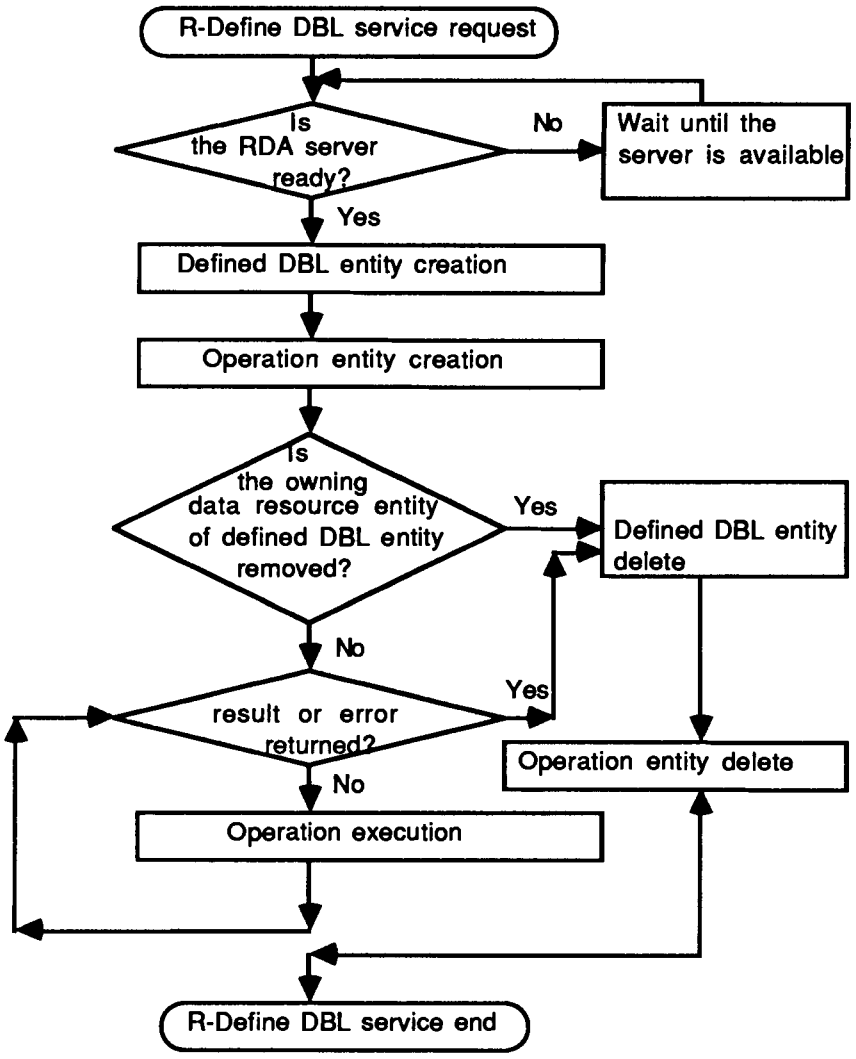


Figure 6-15 R-DefineDBL Service Operation Execution

* R-Invoke Service Request

The R-Invoke service operation execution is depicted in Figure 6-16. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

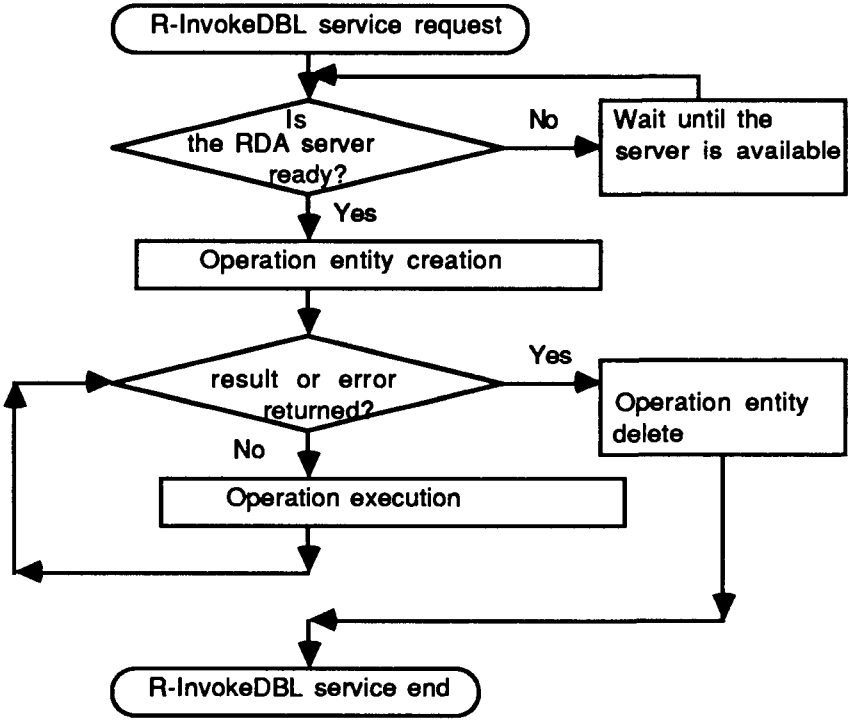


Figure 6-16 R-InvokeDBL Service Operation Execution

* R-DropDBL Service Request

The R-DropDBL service operation execution is depicted in Figure 6-17. The necessary library functions for this operation execution are as follows:

r_operation_entity_create()
r_operation_entity_delete()
r_opened_dr_entity_delete()
r_ddbl_entity_delete()
r_entity_moidify()
r_implementor_error_handle()

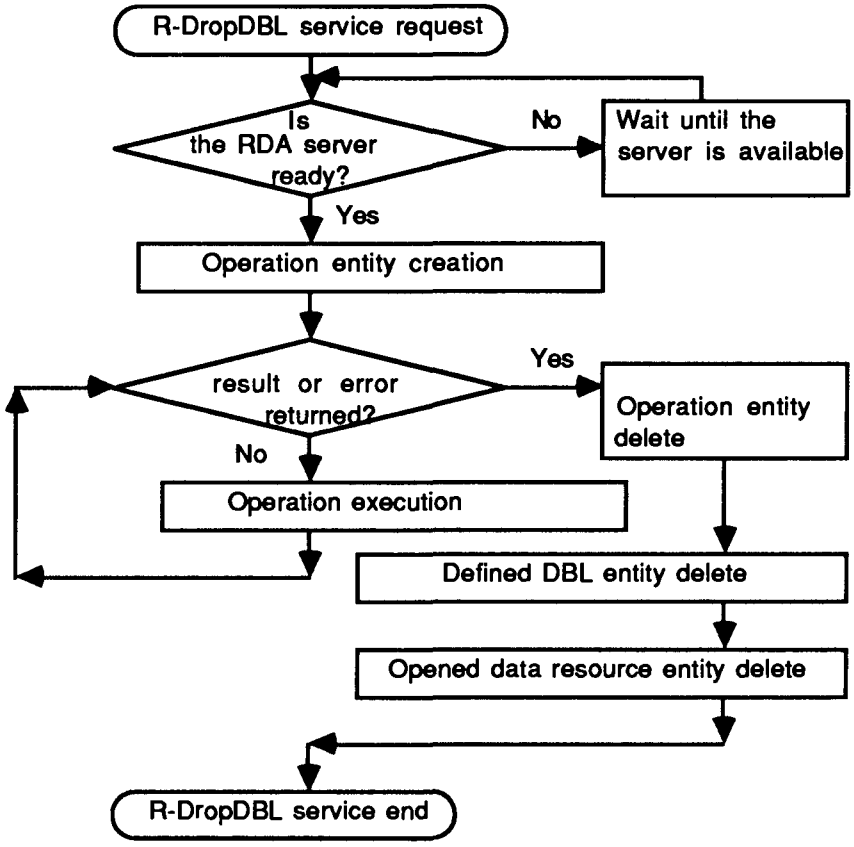


Figure 6-17 R-DropDBL Service Operation Execution

Besides the functions described as the necessary in executing the requested RDA operation for each service, the functions described in the following are required in every service operation execution. In the operation execution step of every service, first the RDA server creates an SQL Server Interface endpoint and returns protocol specific information associated with that point by calling **r_open_sql_sever_endpoint()**. It also returns a file descriptor that serves as the local identifier of the endpoint. There are two input parameters for this function. The first one is r_name which points to an RDA server. The second one is r_info which points to a structure r_info which contains the information about the local SQL Server Interface endpoint. If the function call is successful, a valid file descriptor that identifies a particular SQL Server Interface endpoint is returned.

After opening the SQL server endpoint, the function **r_bind_sql_server_endpoint** () associates a protocol address with a given SQL server endpoint, thereby activating the endpoint. It also directs the RDA Server to begin accepting connect indications if desired. There are three input parameters:

fd, req, and ret. The fd specifies a particular SQL server endpoint. The req is used to request that an address, represented by the predefined structure which contains the protocol address and

connect indication information, be bound to the given SQL Server Interface endpoint. The ret contains the address that the RDA Server actually bound to the SQL Server Interface endpoint; this may be different from the address specified by the user in req. If this function call is successful, a value 0 is returned. Otherwise, a value of -1 is returned.

If all the previous function calls are successful, then for each service request, the function which has the name like r_xxx_yyy_sql_server() where xxx is the service name and yyy is 'send' or 'receive' is called to execute the operation of service requested originally by the RDA client. For example, for the R-Initialize service, the function r_initialize_send_sql_server() is called to send the R-initialize operation to the SQL server and r_initialize_receive_sql_server() is called to receive the result of operation execution from the SQL server. For all the RDA services, the operation execution can be done like this way.

After getting the operation result, the RDA server calls the function **r_unbind_sql_server_endpoint()** which disables an SQL Server Interface endpoint such that no further request destined for the given endpoint will be accepted by the SQL server. The input parameters is the fd which specifies a particular SQL Server Interface endpoint. If this function call is successful, a value 0 is returned. Otherwise, a value of -1 is returned.

The following step is calling the function **r_close_sql_server_endpoint()** to inform the RDA server that the RDA server finished with the SQL server endpoint, and frees any local library resources associated with that endpoint. In addition, this function closes the file associated the the SQL Server Interface endpoint. One input parameter is the fd which specifies a particular SQL server endpoint. If this function call is successful, a value 0 is returned. Otherwise, a value of -1 is returned.

The result of RDA service request executed by the cooperation between the RDA server and SQL server is sent the RDA Server Interface by the function **r_snd_response()** which send the RDA response on the RDA Server endpoint to the RDA Server Interface endpoint by using input parameters the r_server_endpoint that is the endpoint of RDA server and r_si_endpoint that is the endpoint of RDA Server Interface.

After completing the previous steps, the result of RDA service request is sent to the RDA client via the RDA Server Protocol Machine, RDA Communications Service, RDA Client Protocol Machine, and finally through the RDA Client Interface.

CHAPTER VII

Conclusions

As the interworking of computer systems grows, the need for the interconnection of applications and database systems under different manufacturers, managements, complexity levels, and technologies must be satisfied with a minimum of technical agreement outside the interconnection standards. This multi-database system interworking can be achieved by the Remote Database Access (RDA).

The RDA standard is one of a set of International Standards to facilitate the interworking of computer systems. RDA is located in the Application layer of the Open Systems Interconnection (OSI) Basic Reference Model and is related with other International Standards in the OSI set. The two parts of RDA standard, Generic RDA standard and Specialization RDA standard, are now available and they are under development.

Currently, only one database language SQL was specialized. To accommodate the other types of database access, the functionality of RDA must be expanded. The access on the hierarchical and network type database also must be included to be specialized.

The development of RDA standard is slower than the other application protocol standards. The ideas behind the RDA standard must be realized by implementing each actual database language for a particular DB access.

The specialization processes for model, services and protocol, and application contexts were summarized on the database language SQL. Still, some of the parameters must be defined more precisely to clarify the operations of the RDA.

As the basic step of constructing prototype RDA for a subset of database languages, two RDA service user interfaces were designed and necessary functions and parameters were defined. First, to fill the gap of functionalities between the RDA client and the RDA Communications Service, one RDA Client Interface model was suggested and necessary library functions and parameters were defined. Second, an RDA Server Interface model which contains necessary library functions and parameters to send an RDA indication and receive the response for it was designed to fill the gap of functionalities between the RDA server and the RDA Communications Service. The operation scenarios for these two interfaces were explained by using the suggested functions.

Also, a set of Generic Object Management Library functions for the RDA server as one possible implementation model was defined and the functions for the RDA server to interface with SQL Server Interface for the RDA specialization were refined and added. The internal execution of RDA operation according to the RDA server rules were explained by using the functions of Generic Object Management Library for the RDA server.

The set of functions suggested for the RDA Client Interface, RDA Server Interface, RDA server can be a good model in implementing a complete RDA for SQL specialization. All the functions were designed by the object-oriented concept. So, this model can be modified conveniently and implemented easily to accommodate other types of database languages by the object-oriented languages because of functional modularities of library functions.

ASN.1 Module for RDA Application Protocol Data Unit (APDU)

```
ISO9579-RDAxxx { iso standard rda(9579) part(n) module(0) version(1) }
- -
--The "xxx" in "RDAxxx" must be chosen to correspond to the RDA Specialization-
--The "n" in "part(n)" must be the same as that for the RDA Specialization--

DEFINITIONS ::= BEGIN

IMPORTS      AP-title,
             AE-qualifier,
             AP-invocation-identifier,
             AE-invocation-identifier
                    FROM ACSE-1 { joint-iso-ccitt standard acse(8650) } ;

RDA-APDU     ::= CHOICE
       {rda-request-apdu    [0] IMPLICIT RDA-Request-APDU,
          rda-result-apdu   [1] IMPLICIT RDA-Result-APDU,
        rda-error-apdu      [2] IMPLICIT RDA-Error-APDU
       }

RDA-Request_APDU ::= SEQUENCE
       {   operationID              OperationID,
         argument           CHOICE
         {r-Initialize-arg          [0] IMPLICIT R-Initialize-Argument,
           r-Terminate-arg          [1] IMPLICIT R-Terminate-Argument,
           r-BeginTransaction-arg   [2] IMPLICIT NULL,
           r-Commit-arg             [3] IMPLICIT NULL,
           r-Rollback-arg           [4] IMPLICIT NULL,
           r-Cancel-arg             [5] IMPLICIT R-Cancel-Argument,
           r-Status-arg             [6] IMPLICIT R-Status-Argument,
           r-Open-arg               [7] IMPLICIT R-Open-Argument,
           r-Clase-arg              [8] IMPLICIT R-Close-Argument,
           r-ExecuteDBL-arg         [9] IMPLICIT R-EexcuteDBL-Argument,
           r-DefineDBL-arg          [10] IMPLICIT R-DefineDBL-Argument,
           r-InvokeDBL-arg          [11] IMPLICIT R-InvokeDBL-Argument,
           r-DropDBL-arg            [12] IMPLICIT R-DropDBL-Argument
         }
       }

RDA-Result-APDU    ::= SEQUENCE
       {   operationID              OperationID,
         result             CHOICE

         {     r-Initialize-res     [0] IMPLICIT R-Initialize-Result,
           r-Terminate-res          [1] IMPLICIT R-Terminate-Result,
           r-Commit-res             [3] IMPLICIT R-Commit-Result,
           r-Rollback-res           [4] IMPLICIT NULL,
           r-Cancel-res             [5] IMPLICIT R-Cancel-Result,
           r-Status-res             [6] IMPLICIT R-Status-Result,
           r-Open-res               [7] IMPLICIT R-Open-Result,
```

```
        r-Clase-res             [8] IMPLICIT R-Close-Result,
        r-ExecuteDBL-res        [9] IMPLICIT R-EexcuteDBL-Result,
        r-DefineDBL-res         [10] IMPLICIT R-DefineDBL-Result,
        r-InvokeDBL-res         [11] IMPLICIT R-InvokeDBL-Result,
        r-DropDBL-res           [12] IMPLICIT R-DropDBL-Result
        }
      }

RDA-Error-APDU    ::= SEQUENCE
      {  operationID              OperationID,
       error              CHOICE
       {    r-Initialize-err     [0] IMPLICIT R-Initialize-Error,
        r-Terminate-err           [1] IMPLICIT R-Terminate-Error,
        r-BeginTransaction-err    [2] IMPLICIT R-BeginTransaction-Error,
        r-Commit-err              [3] IMPLICIT R-Commit-Error,
        r-Rollback-err            [4] IMPLICIT R-Rollback-Error,
        r-Cancel-err              [5] IMPLICIT R-Cancel-Error,
        r-Status-err              [6] IMPLICIT R-Status-Error,
        r-Open-err                [7] IMPLICIT R-Open-Error,
        r-Clase-err               [8] IMPLICIT R-Close-Error,
        r-ExecuteDBL-err          [9] IMPLICIT R-EexcuteDBL-Error,
        r-DefineDBL-err           [10] IMPLICIT R-DefineDBL-Error,
        r-InvokeDBL-err           [11] IMPLICIT R-InvokeDBL-Error,
        r-DropDBL-err             [12] IMPLICIT R-DropDBL-Error
        }
      }


R-Initialize-Argument       ::= SEQUENCE
      {  dialogueIDSuffix                 [0] DialogueIDSuffix,
       identityOfUser                     [1] IMPLICIT VisibleString,
       userAuthenticationData             [2] AuthenticationData OPTIONAL,
       controlServiceDataRequested        [3] IMPLICIT BOOLEAN DEFAULT FALSE,
       functionalUnitsRequested           [4] IMPLICIT FunctionalUnits,
       specificInitializeArgument         [30] SpecificInitializeArgument OPTIONAL
       }

R-Initialize-Result   ::= SEQUENCE
      {  controlServicedata       [0] IMPLICIT SEQUENCE
       {    controlServicesAllowed      [0] IMPLICIT BOOLEAN DEFAULT TRUE,
        controlAuthenticationdata       [1] AuthenticationData OPTIONAL
        }  OPTIONAL,
        functionalUnitsAllowed   [1] IMPLICIT FunctionalUnits,
        specificInitializeresult [30] SpecificInitializeResult OPTIONAL
        }

R-Initialize-Error   ::= CHOICE
      {accessControlViolation     AccessControlViolation,
       duplicateDialogueID        DuplicateDialogueID,
       invalidSequence            InvalidSequence,
       operationAborted           OperationAborted,
       specificInitializeError    SpecificInitializeError,
       userAuthenticationfailure  UserAuthenticationfailure
```

```
        }


R-Terminate-Argument      ::= SEQUENCE
        {   specificTerminateArgument      [30] SpecificTerminateArgument OPTIONAL
        }

R-Terminate-Result  ::= SEQUENCE
        {   specificTerminateResult  [30] SpecificTerminateresult OPTIONAL
        }

R-Terminate-Error  ::= CHOICE
        {   duplicateOperationID      DuplicateOperationID,
          invalidSequence             Invalidsequence,
          operationAborted            OperationAborted,
          specificTerminateError      SpecificTerminateError
          }


R-BeginTransaction-Error  ::= CHOICE
        {   duplicateOperationID      DuplicateOperationID,
          invalidSequence             Invalidsequence,
          operationAborted            OperationAborted,
          serviceNotNegotiated        ServiceNotNegotiated
          }


R-Commit-Result     ::= SEQUENCE
        {   transactionResult                 [0] IMPLICIT ENUMERATED
          {     committed     (0),
            rolledback        (1)
            }
        }

R-Commit-Error      ::= CHOICE
        {   duplicateOperationID              DuplicateOperationID,
          invalidsequence                     InvalidSequence,
          serviceNotNegotiated                ServiceNotNegotiated
          }


R-Rollback-Error    ::= CHOICE
        {   duplicateOperationID              DuplicateOperationID,
          invalidSequence                     Invalissequence,
          serviceNotNegotiated                ServiceNotNegotiated
          }


R-Cancel-Argument  ::= SEQUENCE
        {   controlledDialogue                [0] IMPLICIT SEQUENCE
          {     dialogueIDClientInvocation    [0] IMPLICIT DialogueIDClientInvocation
OPTIONAL,
            dialogueIDSuffix                  [1] DialogueIDSuffix,
```

```
        controlAuthenticationData       [2] AuthenticationData
        } OPTIONAL,
      operationsReferenced      CHOICE
      {     allOperations               [1] IMPLICIT NULL,
        listOfOperationID               [2] IMPLICIT SEQUENCE OF OperationID
        }
      specificCancelArgument            [30] SpecificCancelArgument OPTIONAL
      }

R-Cancel-Result     ::= SEQUENCE
      {   specificCancelResult          [30]  SpecificCancelResult OPTIONAL
        }

R-Cancel-Error      ::= CHOICE
      {   controlAuthenticationFailure  ControlAuthenticationFailure,
        dialogueIDUnknown               DialogueIDUnknown,
        duplicateOperationID            DuplicateOperationID,
        invalidSequence                 InvalidSequence,
        operationAborted                OperationAborted,
        serviceNotNegotiated            ServiceNotNegotiated,
        specificCancelError             SpecificCancelError
        }


R-Status-Argument   ::= SEQUENCE
      {   controlledDialogue            [0] IMPLICIT SEQUENCE
        {     dialogueIDClientInvocation  [0] IMPLICIT DialogueIDClientInvocation
OPTIONAL,
        dialogueIDSuffix                [1] DialogueIDSuffix,
        controlAuthenticationData       [2] AuthenticationData
        } OPTIONAL,
      operationsReferenced      CHOICE
      {     allOperations               [1] IMPLICIT NULL,
        listOfOperationID               [2] IMPLICIT SEQUENCE OF OperationID
        }
      specificStatusArgument            [30] SpecificStatusArgument OPTIONAL
      }

R-Status-Result     ::= SEQUENCE
      {   listOfStatusInformation       [0] IMPLICIT SEQUENCE OF
                                              StatusInformation,
      specificStatusResult              [30] SpecificStatusResult OPTIONAL
      }

R-Status-Error      ::= CHOICE
      {   controlAuthenticationFailure  ControlAuthenticationFailure,
        dialogueIDUnknown               DialogueIDUnknown,
        duplicateOperationID            DuplicateOperationID,
        invalidSequence                 InvalidSequence
        operationAborted                OperationAborted
        serviceNotNegotiated            ServiceNotNegotiated
        specificStatusError             SpecificStatusError
        }
```

```
R-Open-Argument     ::= SEQUENCE
        {   dataResourceHandle          [0] IMPLICIT DataResourceHandle,
            parentDataResourceHandle    [1] IMPLICIT DataResourceHandle OPTIONAL,
            dataResourceName            [2] IMPLICIT VisibleString OPTIONAL,
            specificAccessControl       [3] SpecificAccessControl OPTIONAL,
            specificUsageMode           [4] SpecificUsageMode OPTIONAL,
            specificOpenArgument        [30] SpecificOpenArgument OPTIONAL
            }

R-Open-Result               ::= SEQUENCE
        {   specificOpenResult          [30] SpecificOpenResult OPTIONAL
            }

R-Open-Error        ::= CHOICE
        {   dataResourceNameNotSpecified    DataResourceNameNotSpecified,
            dataResourceNotAvailable        DataResourceNotAvailable,
            dataResourceOpen                DataResourceOpen,
            dataResourceUnknown             DataResourceUnknown,
            duplicateDataResourceHandle     DuplicateDataResourceHandle,
            duplicateOperationID            DuplicateOperationID,
            invalidSequence                 InvalidSequence,
            operationAborted                OperationAborted,
            operationCancelled              OperationCancelled,
            parentDataResourceHandleUnknown ParentDataResourceHandleUnknown,
            serviceNotNegotiated            ServiceNotNegotiated,
            specificOpenError               SpecificOpenError
            }


R-Close-Argument    ::= SEQUENCE
        {   listOfDataResourceHandle[0] IMPLICIT SEQUENCE OF DataResourceHandle
                                                                    OPTIONAL,
            specificCloseArgument       [30] SpecificCloseArgument OPTIONAL
            }

R-Close-Result      ::= SEQUENCE
        {   listOfCloseExceptions       [0] IMPLICIT SEQUENCE OF CloseException
                                                                    OPTIONAL,
            specificCloseResult         [30] SpecificCloseResult OPTIONAL
            }

R-Close-Error       ::= CHOICE
        {   duplicateOperationID            DuplicateOperationID,
            invalidSequence                 InvalidSequence,
            operationAborted                OperationAborted,
            operationCancelled              OperationCancelled,
            serviceNotNegotiated            ServiceNotNegotiated,
            specificCloseError              SpecificCloseError
            }
```

```
R-ExecuteDBL-Argument     ::= SEQUENCE
        {   dataResourceHandle              [0] IMPLICIT DataResourceHandle
                                                          OPTIONAL,
            specificDBLStatement            [1] SpecificDBLStatement,
            specificDBLArgumentSpecification [2] SpecificDBLArgumentSpecification
                                                          OPTIONAL,
            specificDBLResultSpecification  [3] SpecificDBLResultSpecification
                                                          OPTIONAL,
            dblArguments                    CHOICE
            {   singleargument              [4] IMPLICIT SEQUENCE
              {repetitionCount                [0] IMPLICIT INTEGER DEFAULT 1,
                specificDBLArgumentValues     [1] SpecificDBLArgumentValues
                                                          OPTIONAL
                }
              listOfspecificDBLArgumentValues [5] IMPLICIT SEQUENCE OF
                                                    SpecificDBLArgumentValues
              } OPTIONAL
            }

R-ExecuteDBL-Result       ::= SEQUENCE
        {   specificTerminationCode         [0] SpecificTerminationCode OPTIONAL,
            specificDBLResultSpecification  [1] SpecificDBLResultSpecification
                                                          OPTIONAL,
            listOfResultValues              [2] IMPLICIT SEQUENCE OF ResultValues
            }

R-ExecuteDBL-Error        ::= CHOICE
        {   badRepetitionCount              BadRepetitionCount,
            dataResourceHandleNotspecified  DataResourceHandleNotspecified,
            dataResourceHandleUnknown       DataResourceHandleUnknown,
            duplicateOperationID            DuplicateOperationID,
            invalidSequence                 InvalidSequence,
            noDataResourceAvailable         NoDataResourceAvailable,
            operationAborted                OperationAborted,
            operationCancelled              OperationCancelled,
            serviceNotNegotiated            ServiceNotNegotiated,
            specificExecuteDBLError         SpecificExecuteDBLError,
            transactionRolledBack           TransactionRolledBack
            }


R-DefineDBL-Argument      ::= SEQUENCE
        {   commandHandle                   [0] IMPLICIT CommandHandle,
            dataResourceHandle              [1] IMPLICIT DataResourceHandle
OPTIONAL,
            specificDBLStatement            [2] SpecificDBLStatement,
            specificDBLArgumentSpecification[3] SpecificDBLArgumentSpecification
                                                          OPTIONAL,
            specificDBLResultSpecification  [4] SpecificDBLResultSpecification
                                                          OPTIONAL
            }

R-DefineDBL-Result ::= SEQUENCE
```

```
        {   specificDBLResultSpecification  [0]  SpecificDBLResultSpecification
OPTIONAL,
          specificDBLException              [1] SpecificDBLException OPTIONAL
          }

R-DefineDBL-Error ::= CHOICE
        {   dataResourceHandleNotspecified  DataResourceHandleNotspecified,
          dataResourceHandleUnknown         DataResourceHandleUnknown,
          duplicateCommandHandle            DuplicateCommandHandle,
          duplicateOperationID              DuplicateOperationID,
          invalidSequence                   InvalidSequence,
          noDataResourceAvailable           NoDataResourceAvailable,
          operationAborted                  OperationAborted,
          operationCancelled                OperationCancelled,
          serviceNotNegotiated              ServiceNotNegotiated,
          specificDefineDBLError            SpecificDefineDBLError,
          }


R-InvokeDBL-Argument      ::= SEQUENCE
        {   commandHandle                   [0] IMPLICIT CommandHandle,
          dblArguments                        CHOICE
          {   singleargument                [1] IMPLICIT SEQUENCE
            {repetitionCount                  [0] IMPLICIT INTEGER DEFAULT 1,
              specificDBLArgumentValues       [1]  SpecificDBLArgumentValues
OPTIONAL
            }
            listOfspecificDBLArgumentValues [2] IMPLICIT SEQUENCE OF
                                                  SpecificDBLArgumentValues
          } OPTIONAL
          }

R-InvokeDBL-Result        ::= SEQUENCE
        {specificDBLResultSpecification     [0]  SpecificDBLResultSpecification
                                                                  OPTIONAL,
        specificTerminationCode             [1] SpecificTerminationCode OPTIONAL,
          listOfResultValues                [2] IMPLICIT SEQUENCE OF ResultValues
          }

R-InvokeDBL-Error ::= CHOICE
        {   badRepetitionCount              BadRepetitionCount,
          commandhandleUnknown              commandhandleUnknown,
          duplicateOperationID              DuplicateOperationID,
          invalidSequence                   InvalidSequence,
          noDataResourceAvailable           NoDataResourceAvailable,
          operationAborted                  OperationAborted,
          operationCancelled                OperationCancelled,
          serviceNotNegotiated              ServiceNotNegotiated,
          specificInvokeDBLError            SpecificInvokeDBLError,
          transactionRolledBack             TransactionRolledBack
          }
```

```
R-DropDBL-Argument        ::= SEQUENCE
        {   listOfCommandHandle          [0] IMPLICIT SEQUENCE OF CommandHandle
                                                                  OPTIONAL,
            specificDropDBLArgument      [30] SpecificDropDBLArguments OPTIONAL
            }

R-DropDBL-Result    ::= SEQUENCE
        {   listOfDropExceptions         [0] IMPLICIT SEQUENCE OF DropDBLResult
                                                                  OPTIONAL,
            specificdropDBLResult        [30] specificdropDBLresult OPTIONAL
            }

R-DropDBL-Error     ::= CHOICE
        {   duplicateOperationID         DuplicateOperationID,
            invalidSequence              InvalidSequence,
            operationAborted             OperationAborted,
            operationCancelled           OperationCancelled,
            serviceNotNegotiated         ServiceNotNegotiated,
            specificDropDBLError         SpecificDropDBLError,
            }


DialogueIDSuffix        ::= CHOICE
        {                                  [0] IMPLICIT OCTET STRING
            }

AuthenticationData      ::= CHOICE
        {   cstring                      [0] IMPLICIT IA5String,
            ostring                      [1] IMPLICIT OCTET STRING,
            bstring                      [2] IMPLICIT BIT STRING
            }

FunctionalUnits         ::= BIT STRING
        {   termination                  (0),
            transaction                  (1),
            cancel                       (2),
            status                       (3),
            resource                     (4),
            immediate-DBL                (5),
            stored-DBL                   (6)
            }

CommandHandle           ::= INTEGER

DataResourceHandle      ::= INTEGER

OperationID             ::= INTEGER

DialogueIDClientInvocation    ::= SEQUENCE
        {   apTitle                      [0]  AP-title,
            apQualifier                  [1]  AE-qualifier,
            apInvocationID               [2]  AP-invocation-identifier,
            aeInvocationID               [3]  AE-invocation-identifier
```

```
        }

StatusInformation     ::= SEQUENCE
        {   operationID                     [0] IMPLICIT OperationID,
            operationStatus      CHOICE
            {   operationIDUnknown          [1] IMPLICIT NULL,
                awaitingExecution           [2] IMPLICIT NULL,
                executing                   [3] IMPLICIT NULL,
                finished                    [4] IMPLICIT NULL,
                cancelled                   [5] IMPLICIT NULL,
                aborted                     [6]  IMPLICIT VisibleString
                },
            specificOperationStatusResult   [30] SpecificOperationStatusResult
OPTIONAL
            }

CloseException        ::= SEQUENCE
        {   dataResourceHandle              [0] IMPLICIT DataResourceHandle,
            closeException       CHOICE
            {   dataResourceHandleUnknown   [1] IMPLICIT NULL,
                specificCloseException      [30] SpecificCloseException
                }
            }

ResultValue  ::= SEQUENCE
        {   specificDBLException            [0] SpecificDBLException,
            specificDBLResultValues         [1] SpecificDBLResultValues OPTIONAL
            }

DropDBLException     ::= SEQUENCE
        {   commandHandle                   [0] IMPLICIT CommandHandle,
            dropDBLException   CHOICE
            {   commandHandleUnknown        [1] IMPLICIT NULL
                specificDropDBLException    [30] SpecificDropDBLException
                }
            }

--  ============================================  --
-- Definitions of Errors                          --
-- Application Tags 0-99 are reserved for Generic RDA errors  --
--  ============================================  --

AccessControlViolation                  ::= [APPLICATION 0] IMPLICIT NULL
BadRepetitionCount                      ::= [APPLICATION 1] IMPLICIT NULL
CommandHandleUnknown                    ::= [APPLICATION 2] IMPLICIT NULL
ControlAuthenticationFailure            ::= [APPLICATION 3] IMPLICIT NULL
DataResourceHandleNotSpecified          ::= [APPLICATION 4] IMPLICIT NULL
DataresourceHandleUnknown               ::= [APPLICATION 5] IMPLICIT NULL
DataResourceNameNotSpecified            ::= [APPLICATION 6] IMPLICIT NULL
DataResourceNotAvailable                ::= [APPLICATION 7] IMPLICIT NULL
DataResourceOpen                        ::= [APPLICATION 8] IMPLICIT NULL
DataResourceUnknown                     ::= [APPLICATION 9] IMPLICIT NULL
DialogueIDUnknown                       ::= [APPLICATION 10] IMPLICIT NULL
```

```
DuplicateCommandHandle                  ::= [APPLICATION 11] IMPLICIT NULL
DuplicateDataResourceHandle             ::= [APPLICATION 12] IMPLICIT NULL
DuplicateDialogueID                     ::= [APPLICATION 13] IMPLICIT NULL
DuplicateOperationID                    ::= [APPLICATION 14] IMPLICIT NULL
InvalidSequence                         ::= [APPLICATION 15] IMPLICIT SEQUENCE
        {   diagnosticInformation       [0] IMPLICIT ENUMERATED
          {   dialogueNotActive         (1),
            r-InitializeResponsePending (2),
            transactionNotOpen          (3),
            transactionOpen             (4),
            r-CommitResponsePending     (5),
            r-RollbackResponsePending   (6),
            r-TerminateResponsePending  (7),
            dialogueAlreadyActive       (8)
            } OPTIONAL
        }

NoDataresourceAvailable                 ::= [APPLICATION 16] IMPLICIT NULL
OperationAborted            :           ::= [APPLICATION 17] IMPLICIT SEQUENCE
        {   errorType                       [0] IMPLICIT ENUMERATED
          {   transient                 (0),
            permanent                   (1)
          } DEFAULT transient,
          diagnosticInformation         [1] IMPLICIT VisibleString OPTIONAL
          }

OperationCancelled                      ::= [APPLICATION 18] IMPLICIT NULL
ParentDataResourceHandleUnknown         ::= [APPLICATION 19] IMPLICIT NULL
ServiceNotNegotiated                    ::= [APPLICATION 20] IMPLICIT NULL
TransactionRolledBack                   ::= [APPLICATION 21] IMPLICIT NULL
UserAuthenticationFailure               ::= [APPLICATION 22] IMPLICIT NULL

--  ========================================================  --
-- Specialization defined parameters
- -
--  ========================================================  --
-- The definition of the following "specific" types is left to the RDA Specialization.  --
-- The Specialization may choose to replace the prefix "specific" in type and value    --
 -- references with something more meaningful.  If a parameter referencing one of     --
-- these "specific" types is declared OPTIONAL, then that parameter may be deleted     --
-- if it's not needed by the Specialization, or the keyword OPTIONAL may be deleted    --
-- to make that parameter mandatory.                                                   --
--  ========================================================  --
- -
- -     SpecificAccessControl,
- -     SpecificCancelArgument,
- -     SpecificCancelResult,
- -     SpecificCloseArgument,
- -     SpecificCloseException,
- -     SpecificCloseResult,
- -     SpecificDBLArgumentValues,
- -     SpecificDBLArgumentSpecification,
- -     SpecificDBLException,
```

```
- -     SpecificDBLResultSpecification,
- -     SpecificDBLResultValues,
- -     SpecificDBLStatement,
- -     SpecificDropDBLArgument,
- -     SpecificDropDBLException,
- -     SpecificDropDBLResult,
- -     SpecificInitializeArgument,
- -     SpecificInitializeresult,
- -     SpecificOpenArgument,
- -     SpecificOpenResult,
--     SpecificOperationStatusResult,
- -     SpecificStatusArgument,
- -     SpecificStatusResult,
- -     SpecificTerminateArgument,
- -     SpecificTerminateResult,
- -     SpecificTerminationCode,
- -     SpecificUsageMode

--  ========================================================  --
-- Specialization defined errors.
- -
--  ========================================================  --
-- The definition of the following "specific" errors is left to the RDA Specialization.  --
-- A Specialization may choose to replace the prefix "specific" in type and value        --
-- references with something more meaningful.  A Specialization shall define these      --
-- errors in a manner consistent with the Generic RDA errors defined above.  The        --
-- Application tags 100 and above can be used for these errors.  A Specialization may   --
-- choose to delete the "specific"  errors that are not needed.                          --
--  ========================================================  --
- -
- -     SpecificCancelError,
- -     SpecificCloseError,
- -     SpecificDefineDBLError,
- -     SpecificDropDBLError,
- -     SpecificExecuteDBLError,
- -     SpecificInitializeError,
- -     SpecificInvokeDBLError,
- -     SpecificOpenError,
- -     SpecificStatusError,
- -     SpecificTerminateError

END

--  ========================================================  --
 -- RDA Specialization ASN.1 Module Template
--  ========================================================  --
```

ASN.1 Module for RDA SQL Specialization ASE

```
ISO9579-RDASQL { iso standard rda(9579) part(2) module(0)
version(1) }

- -
--  ================================================  --
- -
DEFINITIONS ::= BEGIN
- -
--  ================================================  --
- -
IMPORTS
- -
            AP-title,
            AE-qualifier,
            AP-invocation-identifier,
            AE-invocation-identifier
            FROM ACSE-1 { joint-iso-ccitt standard acse(8650) } ;

--  ================================================  --
-- RDA-SQL APDU's
- -

RDA-APDU    ::= CHOICE
      { rda-request-apdu  [0] IMPLICIT RDA-Request-APDU,
        rda-result-apdu   [1] IMPLICIT RDA-Result-APDU,
       rda-error-apdu     [2] IMPLICIT RDA-Error-APDU
      }

RDA-Request-APDU ::= SEQUENCE
      {  operationID              OperationID,
       argument          CHOICE
       {    r-Initialize-arg     [0] IMPLICIT R-Initialize-Argument,
         r-Terminate-arg         [1] IMPLICIT R-Terminate-Argument,
         r-BeginTransaction-arg  [2] IMPLICIT NULL,
         r-Commit-arg            [3] IMPLICIT NULL,
         r-Rollback-arg          [4] IMPLICIT NULL,
         r-Cancel-arg            [5] IMPLICIT R-Cancel-Argument,
         r-Status-arg            [6] IMPLICIT R-Status-Argument,
         r-Open-arg              [7] IMPLICIT R-Open-Argument,
         r-Clase-arg             [8] IMPLICIT R-Close-Argument,
         r-ExecuteDBL-arg        [9] IMPLICIT R-EexcuteDBL-Argument,
         r-DefineDBL-arg         [10] IMPLICIT R-DefineDBL-Argument,
         r-InvokeDBL-arg         [11] IMPLICIT R-InvokeDBL-Argument,
         r-DropDBL-arg           [12] IMPLICIT R-DropDBL-Argument
       }
      }

RDA-Result-APDU    ::= SEQUENCE
```

```
    {   operationID             OperationID,
        result          CHOICE
        {     r-Initialize-res      [0]  IMPLICIT  R-Initialize-Result,
          r-Terminate-res           [1]  IMPLICIT  R-Terminate-Result,
          r-Commit-res              [3]  IMPLICIT  R-Commit-Result,
          r-Rollback-res            [4]  IMPLICIT  NULL,
          r-Cancel-res              [5]  IMPLICIT  R-Cancel-Result,
          r-Status-res              [6]  IMPLICIT  R-Status-Result,
          r-Open-res                [7]  IMPLICIT  R-Open-Result,
          r-Clase-res               [8]  IMPLICIT  R-Close-Result,
          r-ExecuteDBL-res          [9]  IMPLICIT  R-EexcuteDBL-Result,
          r-DefineDBL-res           [10]  IMPLICIT  R-DefineDBL-Result,
          r-InvokeDBL-res           [11]  IMPLICIT  R-InvokeDBL-Result,
          r-DropDBL-res             [12]  IMPLICIT  R-DropDBL-Result
          }
        }

RDA-Error-APDU    ::=SEQUENCE
    {   operationID             OperationID,
        error           CHOICE
        {     r-Initialize-err      [0]  IMPLICIT  R-Initialize-Error,
          r-Terminate-err           [1]  IMPLICIT  R-Terminate-Error,
          r-BeginTransaction-err    [2]  IMPLICIT  R-BeginTransaction-Error,
          r-Commit-err              [3]  IMPLICIT  R-Commit-Error,
          r-Rollback-err            [4]  IMPLICIT  R-Rollback-Error,
          r-Cancel-err              [5]  IMPLICIT  R-Cancel-Error,
          r-Status-err              [6]  IMPLICIT  R-Status-Error,
          r-Open-err                [7]  IMPLICIT  R-Open-Error,
          r-Clase-err               [8]  IMPLICIT  R-Close-Error,
          r-ExecuteDBL-err          [9]  IMPLICIT  R-EexcuteDBL-Error,
          r-DefineDBL-err           [10]  IMPLICIT  R-DefineDBL-Error,
          r-InvokeDBL-err           [11]  IMPLICIT  R-InvokeDBL-Error,
          r-DropDBL-err             [12]  IMPLICIT  R-DropDBL-Error
          }
        }

--  ============================================================

R-Initialize-Argument       ::=SEQUENCE
    {   dialogueIDSuffix                [0]  DialogueIDSuffix,
        identityOfUser                  [1]  IMPLICIT  VisibleString,
        userAuthenticationData          [2] AuthenticationData OPTIONAL,
        controlServiceDataRequested     [3] IMPLICIT BOOLEAN DEFAULT FALSE,
        functionalUnitsRequested        [4]  IMPLICIT  FunctionalUnits,
        sQLInitializeArgument           [30]   SQLInitializeArgument
                                               OPTIONAL
        }

R-Initialize-Result   ::=SEQUENCE
    {   controlServicedata      [0] IMPLICIT SEQUENCE
        {     controlServicesAllowed      [0] IMPLICIT BOOLEAN DEFAULT TRUE,
          controlAuthenticationdata       [1] AuthenticationData OPTIONAL
          }   OPTIONAL,
```

```
        functionalUnitsAllowed   [1] IMPLICIT FunctionalUnits,
        sQLInitializeResult      [30] SQLInitializeResult OPTIONAL
        }

R-Initialize-Error  ::= CHOICE
      {  accessControlViolation   AccessControlViolation,
        duplicateDialogueID          DuplicateDialogueID,
        invalidSequence           InvalidSequence,
        operationAborted          OperationAborted,
        sQLInitializeError        SQLnitializeError,
        userAuthenticationfailure UserAuthenticationfailure
        }

--  ============================================================

R-Terminate-Argument      ::= NULL

R-Terminate-Result        ::= NULL

R-Terminate-Error  ::= CHOICE
      {  duplicateOperationID     DuplicateOperationID,
        invalidSequence           Invalidsequence,
        operationAborted          OperationAborted,
        }

--  ============================================================

R-BeginTransaction-Error  ::= CHOICE
      {  duplicateOperationID     DuplicateOperationID,
        invalidSequence           Invalidsequence,
        operationAborted          OperationAborted,
        serviceNotNegotiated           ServiceNotNegotiated
        }

--  ============================================================

R-Commit-Result    ::= SEQUENCE
      {  transactionResult              [0] IMPLICIT ENUMERATED
        {    committed     (0),
        rolledback        (1)
        }
      }

R-Commit-Error     ::= CHOICE
      {  duplicateOperationID           DuplicateOperationID,
        invalidsequence                 InvalidSequence,
        serviceNotNegotiated            ServiceNotNegotiated
        }

- -
============================================================

R-Rollback-Error   ::= CHOICE
```

```
        {   duplicateOperationID            DuplicateOperationID,
            invalidSequence                 Invalissequence,
            serviceNotNegotiated            ServiceNotNegotiated
            }

- -
================================================================

R-Cancel-Argument                   ::= SEQUENCE
        {   controlledDialogue      [0] IMPLICIT SEQUENCE
            {dialogueIDClientInvocation         [0] IMPLICIT DialogueIDClientInvocation
                                                    OPTIONAL,
              dialogueIDSuffix                  [1]  DialogueIDSuffix,
              controlAuthenticationData         [2]  AuthenticationData
              } OPTIONAL,
            operationsReferenced    CHOICE
            {allOperations          [1] IMPLICIT NULL,
              listOfOperationID     [2] IMPLICIT SEQUENCE OF OperationID
              }
            }

R-Cancel-Result     ::= NULL

R-Cancel-Error      ::= CHOICE
        {   controlAuthenticationFailure    ControlAuthenticationFailure,
            dialogueIDUnknown               DialogueIDUnknown,
            duplicateOperationID            DuplicateOperationID,
            invalidSequence                 InvalidSequence,
            operationAborted                OperationAborted,
            serviceNotNegotiated            ServiceNotNegotiated,
            }

--  ============================================================

R-Status-Argument                   ::= SEQUENCE
        {   controlledDialogue      [0] IMPLICIT SEQUENCE
            {dialogueIDClientInvocation         [0] IMPLICIT DialogueIDClientInvocation
                                                    OPTIONAL,
              dialogueIDSuffix                  [1]  DialogueIDSuffix,
              controlAuthenticationData         [2]  AuthenticationData
              } OPTIONAL,
            operationsReferenced    CHOICE
            {allOperations          [1] IMPLICIT NULL,
              listOfOperationID     [2] IMPLICIT SEQUENCE OF OperationID
              }
            }

R-Status-Result     ::= SEQUENCE
        {   listOfStatusInformation [0] IMPLICIT SEQUENCE OF StatusInformation
            }

R-Status-Error      ::= CHOICE
        {   controlAuthenticationFailure    ControlAuthenticationFailure,
```

```
        dialogueIDUnknown                DialogueIDUnknown,
        duplicateOperationID             DuplicateOperationID,
        invalidSequence                  InvalidSequence
        operationAborted                 OperationAborted
        serviceNotNegotiated             ServiceNotNegotiated
        }

--  ================================================================

R-Open-Argument    ::= SEQUENCE
      {  dataResourceHandle        [0] IMPLICIT DataResourceHandle,
        parentDataResourceHandle  [1] IMPLICIT DataResourceHandle OPTIONAL,
        dataResourceName          [2] IMPLICIT VisibleString OPTIONAL,
        sQLAccessControl          [3] AuthenticationData OPTIONAL,
        sQLUsageMode              [4] SQLUsageMode OPTIONAL,
        sQLOpenArgument           [30] SQLOpenArgument OPTIONAL
        }

R-Open-Result              ::= SEQUENCE
      {  sQLOpenResult            [30] SQLOpenResult OPTIONAL
        }

R-Open-Error       ::= CHOICE
      {  dataResourceNameNotSpecified   DataResourceNameNotSpecified,
        dataResourceNotAvailable        DataResourceNotAvailable,
        dataResourceOpen                DataResourceOpen,
        dataResourceUnknown             DataResourceUnknown,
        duplicateDataResourceHandle     DuplicateDataResourceHandle,
        duplicateOperationID            DuplicateOperationID,
        invalidSequence                 InvalidSequence,
        operationAborted                OperationAborted,
        operationCancelled              OperationCancelled,
        parentDataResourceHandleUnknown ParentDataResourceHandleUnknown,
        serviceNotNegotiated            ServiceNotNegotiated,
        sQLOpenError                    SQLOpenError
        }

--  ================================================================

R-Close-Argument   ::= SEQUENCE
      {  listOfDataResourceHandle       [0] IMPLICIT SEQUENCE OF
DataResourceHandle
          OPTIONAL
        }

R-Close-Result     ::= SEQUENCE
      {  listOfCloseExceptions          [0] IMPLICIT SEQUENCE OF CloseException
OPTIONAL
        }

R-Close-Error      ::= CHOICE
      {  duplicateOperationID           DuplicateOperationID,
        invalidSequence                 InvalidSequence,
```

```
        operationAborted                OperationAborted,
        operationCancelled              OperationCancelled,
        serviceNotNegotiated            ServiceNotNegotiated,
        }

--  ================================================================

R-ExecuteDBL-Argument    ::= SEQUENCE
      {dataResourceHandle          [0] IMPLICIT DataResourceHandle OPTIONAL,
      sQLDBLStatement                    [1]  SQLDBLStatement,
      sQLDBLArgumentSpecification  [2]  SQLDBLArgumentSpecification
                                                    OPTIONAL,
      sQLDBLResultSpecification          [3]  SQLDBLResultSpecification
                                                    OPTIONAL,
      dblArguments                       CHOICE
        {   singleargument               [4] IMPLICIT SEQUENCE
          {repetitionCount                 [0] IMPLICIT INTEGER DEFAULT 1,
          sQLDBLArgumentValues             [1]  SQLDBLArgumentValues
                                                    OPTIONAL
            }
          listOfSQLDBLArgumentValues       [5] IMPLICIT SEQUENCE OF
                                           SQLDBLArgumentValues
          } OPTIONAL
        }

R-ExecuteDBL-Result      ::= SEQUENCE
      {   sQLTerminationCode             [0]  SpecificTerminationCode
                                                    OPTIONAL,
        specificDBLResultSpecification   [1] SQLDBLResultSpecification OPTIONAL,
        listOfResultValues               [2] IMPLICIT SEQUENCE OF ResultValues
        }

R-ExecuteDBL-Error       ::= CHOICE
      {   badRepetitionCount             BadRepetitionCount,
        dataResourceHandleNotspecified   DataResourceHandleNotspecified,
        dataResourceHandleUnknown        DataResourceHandleUnknown,
        duplicateOperationID             DuplicateOperationID,
        invalidSequence                  InvalidSequence,
        noDataResourceAvailable          NoDataResourceAvailable,
        operationAborted                 OperationAborted,
        operationCancelled               OperationCancelled,
        serviceNotNegotiated             ServiceNotNegotiated,
        sQLExecuteDBLError               SQLExecuteDBLError,
        transactionRolledBack            TransactionRolledBack
        }

--  ================================================================

R-DefineDBL-Argument     ::= SEQUENCE
      {   commandHandle                        [0] IMPLICIT CommandHandle,
        dataResourceHandle                     [1] IMPLICIT DataResourceHandle
OPTIONAL,
      sQLDBLStatement                    [2]  SQLDBLStatement,
```

```
      sQLDBLArgumentSpecification [3]
                                  SQLDBLArgumentSpecification
                                                OPTIONAL,
       sQLDBLResultSpecification  [4] SQLDBLResultSpecification
                                                OPTIONAL
       }

R-DefineDBL-Result ::= SEQUENCE
      {sQLDBLResultSpecification     [0] SQLDBLResultSpecification
                                                OPTIONAL,
       sQLDBLException               [1] SQLDBLException OPTIONAL
       }

R-DefineDBL-Error ::= CHOICE
      {  dataResourceHandleNotspecified DataResourceHandleNotspecified,
       dataResourceHandleUnknown      DataResourceHandleUnknown,
       duplicateCommandHandle         DuplicateCommandHandle,
       duplicateOperationID           DuplicateOperationID,
       invalidSequence                InvalidSequence,
       noDataResourceAvailable        NoDataResourceAvailable,
       operationAborted               OperationAborted,
       operationCancelled             OperationCancelled,
       serviceNotNegotiated           ServiceNotNegotiated,
       sQLDefineDBLError              SQLDefineDBLError,
       }

--  ============================================================

R-InvokeDBL-Argument     ::= SEQUENCE
      {  commandHandle                [0] IMPLICIT CommandHandle,
       dblArguments                   CHOICE
       {  singleargument              [1] IMPLICIT SEQUENCE
         {repetitionCount               [0] IMPLICIT INTEGER DEFAULT 1,
           sQLDBLArgumentValues         [1] SQLDBLArgumentValues
                                                OPTIONAL
         }
         listOfSQLDBLArgumentValues [2] IMPLICIT SEQUENCE OF
                                          SQLDBLArgumentValues
         } OPTIONAL
       }

R-InvokeDBL-Result       ::= SEQUENCE
      {  sQLDBLResultSpecification [0] SQLDBLResultSpecification
                                                OPTIONAL,
       listOfResultValues             [2] IMPLICIT SEQUENCE OF ResultValues
       }

R-InvokeDBL-Error ::= CHOICE
      {  badRepetitionCount           BadRepetitionCount,
       commandHandleUnknown           commandhandleUnknown,
       duplicateOperationID           DuplicateOperationID,
       invalidSequence                InvalidSequence,
       noDataResourceAvailable        NoDataResourceAvailable,
```

```
        operationAborted                OperationAborted,
        operationCancelled              OperationCancelled,
        serviceNotNegotiated            ServiceNotNegotiated,
        sQLInvokeDBLError               SQLInvokeDBLError,
        transactionRolledBack           TransactionRolledBack
        }

--  ============================================================

R-DropDBL-Argument      ::= SEQUENCE
        {   listOfCommandHandle         [0] IMPLICIT SEQUENCE OF CommandHandle
                                                        OPTIONAL,
        }

R-DropDBL-Result    ::= SEQUENCE
        {   listOfDropExceptions            [0] IMPLICIT SEQUENCE OF
DropDBLResult
            OPTIONAL,

R-DropDBL-Error     ::= CHOICE
        {   duplicateOperationID        DuplicateOperationID,
          invalidSequence               InvalidSequence,
          operationAborted              OperationAborted,
          operationCancelled            OperationCancelled,
          serviceNotNegotiated          ServiceNotNegotiated,
          }

--  ============================================================

DialogueIDSuffix        ::= CHOICE
        {                               [0] IMPLICIT OCTET STRING
          }

AuthenticationData      ::= CHOICE
        {   cstring                     [0]  IMPLICIT  IA5String,
          ostring                       [1] IMPLICIT OCTET STRING,
          bstring                       [2] IMPLICIT BIT STRING
          }

FunctionalUnits         ::= BIT STRING
        {   termination                 (0),
          transaction                   (1),
          cancel                        (2),
          status                        (3),
          resource                      (4),
          immediate-DBL                 (5),
          stored-DBL                    (6)
          }

CommandHandle           ::= INTEGER

DataResourceHandle      ::= INTEGER
```

```
OperationID             ::= INTEGER

DialogueIDClientInvocation   ::= SEQUENCE
        {   apTitle                         [0]  AP-title,
            apQualifier                     [1]  AE-qualifier,
            apInvocationID                  [2]  AP-invocation-identifier,
            aeInvocationID                  [3]  AE-invocation-identifier
        }

StatusInformation       ::= SEQUENCE
        {   operationID                     [0] IMPLICIT OperationID,
            operationStatus     CHOICE
            {operationIDUnknown             [1] IMPLICIT NULL,
              awaitingExecution             [2] IMPLICIT NULL,
              executing                     [3] IMPLICIT NULL,
              finished                      [4] IMPLICIT NULL,
              cancelled                     [5] IMPLICIT NULL,
              aborted                       [6]  IMPLICIT VisibleString
              }
            }

CloseException          ::= SEQUENCE
        {   dataResourceHandle              [0] IMPLICIT DataResourceHandle,
            closeException      CHOICE
            {dataResourceHandleUnknown      [1] IMPLICIT NULL,
              }
            }

ResultValue ::= SEQUENCE
        {sQLDBLException                    [0]  SQLDBLException,
            sQLDBLResultValues              [1]  SQLDBLResultValues  OPTIONAL
            }

DropDBLException        ::= SEQUENCE
        {   commandHandle                   [0] IMPLICIT CommandHandle,
            dropDBLException    CHOICE
            {   commandHandleUnknown        [1] IMPLICIT NULL
              }
            }

--  ============================================================
- -
-- Definitions of Errors
- -
-- Application Tags 0-99 are reserved for Generic RDA errors
- -
--  ============================================================
- -

AccessControlViolation                  ::= [APPLICATION 0] IMPLICIT NULL
BadRepetitionCount                      ::= [APPLICATION 1] IMPLICIT NULL
CommandHandleUnknown                    ::= [APPLICATION 2] IMPLICIT NULL
ControlAuthenticationFailure            ::= [APPLICATION 3] IMPLICIT NULL
```

```
DataResourceHandleNotSpecified          ::= [APPLICATION 4] IMPLICIT NULL
DataresourceHandleUnknown               ::= [APPLICATION 5] IMPLICIT NULL
DataResourceNameNotSpecified            ::= [APPLICATION 6] IMPLICIT NULL
DataResourceNotAvailable                ::= [APPLICATION 7] IMPLICIT NULL
DataResourceOpen                        ::= [APPLICATION 8] IMPLICIT NULL
DataResourceUnknown                     ::= [APPLICATION 9] IMPLICIT NULL
DialogueIDUnknown                       ::= [APPLICATION 10] IMPLICIT NULL
DuplicateCommandHandle                  ::= [APPLICATION 11] IMPLICIT NULL
DuplicateDataResourceHandle             ::= [APPLICATION 12] IMPLICIT NULL
DuplicateDialogueID                     ::= [APPLICATION 13] IMPLICIT NULL
DuplicateOperationID                    ::= [APPLICATION 14] IMPLICIT NULL
InvalidSequence                         ::= [APPLICATION 15] IMPLICIT SEQUENCE
      {  diagnosticInformation          [0] IMPLICIT ENUMERATED
        {  dialogueNotActive            (1),
         r-InitializeResponsePending    (2),
         transactionNotOpen             (3),
         transactionOpen                (4),
         r-CommitResponsePending        (5),
         r-RollbackResponsePending      (6),
         r-TerminateResponsePending     (7),
         dialogueAlreadyActive          (8)
         } OPTIONAL
      }

NoDataresourceAvailable                 ::= [APPLICATION 16] IMPLICIT NULL
OperationAborted            :           ::= [APPLICATION 17] IMPLICIT SEQUENCE
      {  errorType                        [0] IMPLICIT ENUMERATED
        {  transient                      (0),
         permanent                        (1)
        } DEFAULT transient,
        diagnosticInformation           [1] IMPLICIT VisibleString OPTIONAL
        }

OperationCancelled                      ::= [APPLICATION 18] IMPLICIT NULL
ParentDataResourceHandleUnknown         ::= [APPLICATION 19] IMPLICIT NULL
ServiceNotNegotiated                    ::= [APPLICATION 20] IMPLICIT NULL
TransactionRolledBack                   ::= [APPLICATION 21] IMPLICIT NULL
UserAuthenticationFailure               ::= [APPLICATION 22] IMPLICIT NULL

--  ============================================================
- -
-- Definition of SQL Specialization defined parameters
--  ============================================================
- -

SQLDataStatement ::=  IA5String

SQLDataTypeDescriptor ::= SEQUENCE
      {  colName                    [1]  IMPLICIT  VisibleString
OPTIONAL,
         varName                    [2]  IMPLICIT  IA5String  OPTIONAL,
         indName                    [3]  IMPLICIT  IA5String  OPTIONAL,
         typeDescriptor               CHOICE
```

```
    {  characterType          [5] IMPLICIT SEQUENCE
     -- SQL Type: character
     {  charSet               OBJECT IDENTIFIER OPTIONAL,
        length                INTEGER,
        fixedLengthEncoding   BOOLEAN
        },
    numericType               [6] IMPLICIT SEQUENCE
     -- SQL Type: numeric
     {  precision             INTEGER,
        scale                 INTEGER
        },
    decimalType               [7] IMPLICIT SEQUENCE
     -- SQL Type: decimal
     {  precision             INTEGER,
        scale                 INTEGER
        },
    integerType               [8] IMPLICIT SEQUENCE
     -- SQL Type: integer
     {  precision             INTEGER,
        precisionBase         ENUMERATED
        {    binary           (0),
             decimal          (1)
             }
        },
    smallIntType              [9] IMPLICIT SEQUENCE
     -- SQL Type: smallint
     {  precision             INTEGER,
        precisionBase         ENUMERATED
        {    binary           (0),
             decimal          (1)
             }
        },

    floatType                   [10] IMPLICIT SEQUENCE
     -- SQL Type: float
     {  mantissaPrecision     INTEGER,
        exponentPrecision     INTEGER
        },
    realType                    [11] IMPLICIT SEQUENCE
     -- SQL Type: real
     {  mantissaPrecision     INTEGER,
        exponentPrecision     INTEGER
        },
    doublePrecisionType         [11] IMPLICIT SEQUENCE
     -- SQL Type: doublePrecision
     {  mantissaPrecision     INTEGER,
        exponentPrecision     INTEGER
        },
  }
}

SQLDBLArgumentValues      ::= SQLValueList
```

```
SQLDBLArgumentSpecification ::= SEQUENCE OF SQLDataTypeDescriptor

SQLDBLException             ::= SEQUENCE
      {  sQLState           [0] IMPLICIT VisibleString OPTIONAL
         sQLCode            [1] IMPLICIT INTEGER OPTIONAL,
         sQLErrorText       [2] IMPLICIT VisibleString OPTIONAL
       }

SQLDBLResultSpecification   ::= SEQUENCE OF SQLDataTypeDescriptor

SQLDBLResultValues          ::= SQLValueList

SQLDBLStatement             ::= CHOICE
      {  sQLDataStatement   [0] IMPLICIT SQLDataStatement,
         sQLSchemaStatement [1] IMPLICIT SQLSchemaStatement
       }

SQLInitializeArgument       ::= SEQUENCE
      {  sQLConformanceLevel      [0] IMPLICIT OBJECT IDENTIFIER,
         userData           [1] IMPLICIT OCTET STRING OPTIONAL
      }

SQLOpenArgument             ::= SEQUENCE
      {  charSet            [0] IMPLICIT OBJECT IDENTIFIER OPTIONAL
       }

SQLOpenResult               ::= SEQUENCE
      {  charSet            [0] IMPLICIT OBJECT IDENTIFIER OPTIONAL,
         charSetNotSupported [1] IMPPLICIT NULL OPTIONAL
       }

SpecificOperationStatusResult

SQLSchemaStatement          ::=  IA5String

SQLUsageMode                ::= ENUMERATED
      {  retrieval          (0),
         update             (1)
       }

SQLValue                          ::= SEQUENCE
      {  dataItem           CHOICE
       {                    characterItem      [0] IMPLICIT OCTET
STRING,
         numericItem          [1] IMPLICIT INTEGER,
         decimalItem          [2] IMPLICIT INTEGER,
         integerItem          [3] IMPLICIT INTEGER,
         smallIntItem         [4] IMPLICIT INTEGER,
         floatItem            [5] IMPLICIT REAL,
         realItem             [6] IMPLICIT REAL,
         doublePrecisionItem [7] IMPLICIT REAL
         }                  OPTIONAL,
       indicator             [30] IMPLICIT INTEGER OPTIONAL
```

```
        }

SQLValueList                    ::= CHOICE
      {  external               EXTERNAL,
         internal               SEQUENCE OF SQLValue
        }

--  ================================================  --
-- Definitions of RDA SQL Specialization Errors.
-- The Application tags 100-199 are reserved for SQL Specialization errors.
- -
--  ================================================  --

HostIdentifierError                 ::= [APPLICATION 100] IMPLICIT NULL

InvalidSQLConformanceLevel          ::= [APPLICATION 101] IMPLICIT NULL

SQLAccessControlViolation           ::= [APPLICATION 102] IMPLICIT NULL

SQLDatabaseResourceAlreadyOpenError ::= [APPLICATION 103] IMPLICIT NULL

SQLDBLTransactionStatementNotAllowed ::= [APPLICATION 104 IMPLICIT NULL

SQLDBLArgumentCountMismatch         ::= [APPLICATION 105] IMPLICIT NULL

SQLDBLArgumentTypeMismatch          ::= [APPLICATION 106] IMPLICIT NULL

SQLUsageModeViolation               ::= [APPLICATION 107] IMPLICIT NULL

RDATransactionNotOpen               ::= [APPLICATION 108] IMPLICIT NULL

- -
-- SQL Specialization errors which can be returned for specific RDA operations
- -

SpecificDefineDBLError    ::=   CHOICE
                                SQLUsageModeViolation,
                                SQLDBLTransactionStatementAllowed,
                                HostIdentifierError

SpecificExecuteDBLError   ::=   CHOICE
                                SQLUsageModeViolation,
                                SQLDBLTransactionStatementNotAllowed,
                                SQLDBLArgumentCountMismatch,
                                SQLDBLArgumentTypeMismatch,
                                HostIdentifierError,
                                RDATransactionNotOpen

SQLInitializeError        ::=   CHOICE
                                InvalidSQLConformanceLevel

SQLInvokeDBLError         ::=   CHOICE
                                SQLUsageModeViolation,
```

```
                                SQLDBLArgumentCountMismatch,
                                SQLDBLArgumentTypeMismatch,
                                RDATransactionNotOpen

SQLOpenError                    ::=     CHOICE
                                SQLAccessControlViolation,
                                SQLUsageModeViolation,
                                SQLdatabaseresourceAlreadyOpenError

--  ================================================================
END -- RDA SQL Specialization  ASN.1 Module
--  ================================================================
```

REFERENCES

[1] C. J. Date. A Guide to the SQL Standard, Second Edition, Addison-Wesley Publishing Company, 1989.

[2] Henshall, J., and Shaw, A. OSI Explained - End to End Computer Communication Standards. Chichester, England, Ellis Horwood, 1988.

[3] Information Systems - Open Systems - Remote Database Access Tutorial. ISO/JTC 1/SC 21 N 3343, January 1989.

[4] Information Processing Systems - Open Systems Interconnection - Remote Database Access - Part 1: Generic Model, Service, and Protocol. ISO/IEC JTC1/SC 21, DP 9579-1, March 1990.

[5] Information Processing Systems - Open Systems Interconnection - Remote Database Access - Part 1: Generic Model, Service, and Protocol. ISO/IEC JTC1/SC 21, DIS 9579-1, August 15, 1991.

[6] Information Processing Systems - Open Systems Interconnection - Remote Database Access - Part 2: SQL Specialization. ISO/IEC JTC1/SC 21, DP 9579-2, February 1990.

[7] Information Processing Systems - Open Systems Interconnection - Remote Database Access - Part 2: SQL Specialization. ISO/IEC JTC1/SC 21, DIS 9579-2, August 23, 1991.

[8] J. D. Ullman. Principles of Database and Knowledge-Base Systems, Vol. 1, Computer Science Press, 1988.

[9] Knightson, Keith G., Knowles T., and Larmouth, J. Standards for Open Systems Interconnection. New York, McGraw-Hill Company, 1987.

[10] Rose, Marshall T. The Open Book - A Practical Perspective on OSI. Englewood Cliffs, New Jersey, Prentice Hall, 1989.

[11] Tanenbaum, Andrew S. Computer Networks. Second Edition, Englewood Cliffs, New Jersey, Prentice Hall, 1988.

[12] X/Open Portability Guide - Networking Services. X/Open Company, Ltd. Englewood Cliffs, New Jersey, Prentice Hall, 1988.

At the author's request, the Curriculum Vitae (CV) on page 140 has been redacted from this digitized copy.