Neda's Implementation of ESRO – Source

Documentation Applies for:
ESRO-FULL-SRC
ESRO-BASE-SRC
ESRO-TEST-SRC

Neda Document Number: 105-102-06

Last Updated: Author unspecified

Doc. Revision: source unspecified

Neda Communications, Inc.

January 27, 1999

Contents

1 Introduction **15**
1.1 About This Product . . . 15
1.2 About This Document . . . 15
1.3 Architecture . . . 16
1.4 Overview of Software . . . 16
1.4.1 Getting the sources . . . 16
1.4.2 Compiling and Building the Software . . . 18

2 ESRO Protocol Engine **19**
2.1 Introduction . . . 19
2.2 Overview and Concepts . . . 19
2.2.1 ESROP Service Features . . . 19
2.2.2 Software Architecture . . . 19
2.2.2.1 Upper Interface . . . 21
2.2.2.2 Lower Interface . . . 21
2.2.2.3 Timer and Data Unit Management Interface . . . 21
2.2.2.4 Network Management Interface . . . 21
2.3 ESRO Service Primitives . . . 21
2.3.1 SAP Management . . . 22
2.3.2 Operation Invocation . . . 22
2.4 Upper Interface Functions . . . 24
2.4.1 SAP Management . . . 24
2.4.1.1 Bind an ESROP-User . . . 24
2.4.1.2 Unbind an ESROP-User . . . 25
2.4.2 ESROP Action Primitives . . . 25
2.4.2.1 ESROS-INVOKE.request . . . 25
2.4.2.2 ESROS-RESULT.request . . . 26
2.4.2.3 ESROS-ERROR.request . . . 26
2.4.3 ESROS Event Primitives . . . 26
2.4.3.1 ESROS-INVOKE.indication . . . 26
2.4.3.2 ESROS-RESULT.indication . . . 27
2.4.3.3 ESROS-ERROR.indication . . . 27
2.4.3.4 ESROS-FAILURE.indication . . . 27
2.4.3.5 ESROS-RESULT.confirm . . . 28
2.4.3.6 ESROS-ERROR.confirm . . . 28
2.5 Lower Interface Functions . . . 28
2.5.1 Connectionless interface . . . 28
2.5.1.1 UDP SAP Bind . . . 28
2.5.1.2 UDP SAP Unbind . . . 29

2.5.2 Action Primitives . . . 29
2.5.3 Event Primitives . . . 29
2.6 Environment Interface . . . 29
2.6.1 Data Unit Management . . . 29
2.6.2 Timer Management . . . 30
2.7 Network Management . . . 30
2.8 Portation Notes . . . 30
2.8.1 Unresolved Symbols . . . 30
2.8.2 Configuration . . . 30
2.8.3 Tracing . . . 30
2.8.4 Differences Between Unix and MS-DOS/Windows Portations . . . 30
2.8.4.1 Multiple Processes vs. Single Process . . . 32
2.8.4.2 Libraries used building MS-DOS ESROS user application in one process with Call-back API 32
2.8.4.3 Scheduler Module (SCH_) . . . 32
2.8.4.4 Timer Module (TMR_) . . . 33
2.8.4.5 FLEX & BISON . . . 33
2.9 Design Overview . . . 33
2.9.1 SAP Management . . . 33
2.9.2 ESROP Finite State Machine . . . 33
2.9.2.1 Events . . . 33
2.9.2.2 Event Processor . . . 37
2.9.2.3 State Information . . . 37
2.9.3 ESROS Timers . . . 38
2.9.4 ESROP PDU Parser . . . 38
2.9.5 ESROP PDU Formatter . . . 38
2.9.6 Operation Invocation . . . 38
2.9.6.1 Invoker ESROP-Provider . . . 38
2.9.6.2 Performer ESROP-Provider . . . 39

3 ESRO API **41**
3.1 ESROS With Function Call API . . . 41
3.1.1 Initialize the Parameters . . . 41
3.1.2 Activate ESROS Service Access Point . . . 41
3.1.3 Deactivate ESROS Service Access Point . . . 42
3.1.4 ESROS Invoke Service Request . . . 42
3.1.5 ESROS Result Service Request . . . 43
3.1.6 ESROS Error Service Request . . . 43
3.1.7 Get an event . . . 44
3.1.8 Sample Code . . . 44
3.1.8.1 invoker.c . . . 44
3.1.8.2 performer.c . . . 45
3.2 ESROS With Callback API . . . 45
3.2.1 Initialize the Parameters . . . 45
3.2.2 Activate ESROS Service Access Point . . . 45
3.2.3 Deactivate ESROS Service Access Point . . . 47
3.2.4 ESROS Invoke Service Request . . . 48
3.2.5 ESROS Result Service Request . . . 48
3.2.6 ESROS Error Service Request . . . 49
3.2.7 Sample Code . . . 49
3.2.7.1 invoksch.c . . . 49
3.2.7.2 perfsch.c . . . 49

4 ESROS Programs **51**
4.1 Introduction . . . 51
4.2 Providers and Users - Unix vs. MS-DOS/Windows . . . 51
4.2.1 Unix . . . 51
4.2.2 MS-DOS/Windows . . . 51
4.3 The ESROS Service Provider . . . 54
4.3.1 Unix . . . 54
4.3.1.1 Running esros under Unix . . . 54
4.3.2 MS-DOS/Windows . . . 54
4.4 Service User Programs . . . 54
4.4.1 ops_xmpl . . . 55
4.4.1.1 Unix . . . 55
4.4.1.2 MS-DOS/Windows . . . 55
4.4.2 stress . . . 55
4.4.2.1 Unix . . . 55
4.4.2.2 MS-DOS/Windows . . . 55
4.4.3 Tester . . . 56
4.4.3.1 Unix . . . 56
4.4.3.2 MS-DOS/Windows . . . 56
4.5 Support Programs . . . 56
4.5.1 esropscop . . . 56
4.6 Running the Test Programs . . . 56
4.6.1 Unix . . . 56
4.6.1.1 ops_xmpl . . . 56
4.6.1.2 stress . . . 57
4.6.1.3 esrossi . . . 57
4.6.2 MS-DOS/Windows . . . 57
4.6.2.1 esros . . . 57
4.6.2.2 ops_xmpl . . . 57
4.6.2.3 stress . . . 57
4.6.2.4 tester . . . 58
4.7 Senarios . . . 58
4.7.1 Logging-Related Commands . . . 58
4.7.1.1 log . . . 58
4.7.1.2 logfile . . . 58
4.7.1.3 quiet . . . 59
4.7.1.4 verbose . . . 59
4.7.2 Invocation-Related Commands . . . 59
4.7.2.1 invoke request . . . 59
4.7.2.2 invoke indication . . . 60
4.7.3 Result-Related Commands . . . 60
4.7.3.1 result request . . . 60
4.7.3.2 result indication . . . 61
4.7.3.3 result confirmation . . . 61
4.7.4 Error-Related Commands . . . 61
4.7.4.1 error request . . . 61
4.7.4.2 error indication . . . 62
4.7.4.3 error confirmation . . . 62
4.7.5 General Commands . . . 63
4.7.5.1 sapbind . . . 63
4.7.5.2 saprelease . . . 63

4.7.5.3 failure indication . . . 63
4.7.5.4 rawevent . . . 64
4.7.5.5 delay . . . 64
4.7.6 Scenario File Manipulation Commands . . . 65
4.7.6.1 include . . . 65
4.7.6.2 path . . . 65
4.8 Tracing . . . 65
4.8.1 OCP Trace Module Tracing . . . 65
4.8.1.1 Run Time Control of TM_ . . . 66
4.8.1.2 TM Output . . . 66
4.8.2 PDU Transaction Logging . . . 66
4.8.2.1 PDU Transaction Log Display . . . 66
4.9 Implementation Notes . . . 67
4.9.1 Differences Between Unix and MS-DOS Portations . . . 67
4.9.1.1 Multiple Processes vs. Single Process . . . 67
4.9.1.2 Scheduler Module (SCH_) . . . 68
4.9.1.3 Timer Module (TMR_) . . . 68
4.9.1.4 FLEX & BISON . . . 68

5 ESROS Testing 69
5.1 Conformance and Interconnection Testing Overview . . . 69
5.2 Abstract Test Methods . . . 69
5.2.1 What Is an ATM? . . . 70
5.2.1.1 The ATM/ATS Relationship . . . 70
5.2.1.2 Applicability . . . 70
5.2.2 Remote Test Method . . . 70
5.2.3 Coordinated Test Method . . . 72
5.2.4 Distributed Test Method . . . 72
5.2.5 Local Test Method . . . 75
5.3 The ESROS Test Tools . . . 75
5.3.1 How the Test Tools Work . . . 75
5.4 Test Objectives . . . 75
5.4.1 Valid sequences of primitives . . . 75
5.4.2 Invalid sequences of primitives . . . 78
5.4.3 Parameter variations on primitives . . . 78
5.4.4 Stress Tests . . . 78
5.4.5 Multiple results . . . 78
5.5 Test Cases . . . 78
5.5.1 Valid sequences of primitives . . . 78
5.5.1.1 1.001 . . . 78
5.5.1.2 1.002 . . . 78
5.5.1.3 1.003 . . . 78
5.5.1.4 1.004 . . . 78
5.5.1.5 1.005 . . . 78
5.5.2 Invalid sequences of primitives . . . 78
5.5.2.1 2.001 . . . 78
5.5.2.2 2.002 . . . 78
5.5.2.3 2.003 . . . 78
5.5.2.4 2.004 . . . 79
5.5.2.5 2.005 . . . 79
5.5.2.6 2.006 . . . 79

5.5.3 Parameter variations on primitives 79
5.5.3.1 3.001 79
5.5.3.2 3.002 79
5.5.3.3 3.003 79
5.5.4 Stress Tests 79
5.5.4.1 4.001 79
5.5.4.2 4.002 79
5.5.4.3 4.003 79
5.5.4.4 4.004 79
5.5.4.5 4.005 79
5.5.4.6 4.006 79
5.5.4.7 4.007 79
5.5.4.8 4.008 80
5.5.4.9 4.009 80
5.5.5 Multiple Results 80
5.5.5.1 5.001 80
5.6 Example 80

A Acronyms **81**

B ESRO API Example Usage **83**
B.1 invoker.c 83
B.2 invoksch.c 83
B.3 performer.c 83
B.4 perfsch.c 83

C ESRO Program man Pages **85**

D Trace Bit Definitions **93**

E Neda PICS for ESROS **95**
E.1 Introduction 95
E.2 Identification 95
E.2.1 Supplier Identification 95
E.2.2 Implementation Information 95
E.3 ESROS 96
E.3.1 ESROS Protocol Summary 96
E.3.2 ESROS Protocol Capabilities 96
E.3.2.1 Overview of ESRO Services 96
E.3.2.2 Connectionless Oriented Operation 96
E.3.2.3 Segmentation and Reassembly 96
E.3.2.4 Connection Oriented Operation 97
E.3.2.5 Concatenation and Separation 97

List of Tables

2.1 ESRO Service Primitives 22
2.2 ESROS-SAP Management 22
2.3 Service Primitives and corresponding functions 24
2.4 Network Management Counters 30
2.5 Unresolved Symbols of ESROP 31
2.6 ESROP Configuration Parameters 31
2.7 Timers 38

D.1 Complete List of Trace Bit Definitions 94

List of Figures

1.1 Implementation Architecture 17

2.1 Neda's ESROP Module and its Interfaces 20
2.2 Time sequence diagram for ESRO Services 23
2.3 ESROP Modules 34
2.4 ESROP Finite State Machine 35

3.1 Example of time sequence diagram for ESROS Services 46
3.2 Example of time sequence diagram for ESROS CB Services 50

4.1 Unix ESROS Processes. 52
4.2 MS-DOS/Windows ESROS Processes 53

5.1 Remote Test Method 71
5.2 Coordinated Test Method 73
5.3 Distributed Test Method 74
5.4 Local Test Method 76
5.5 ESROS Test Tools 77

Preface

This document provides details on the programming development environment of Neda's source code of ESROS (Efficient Short Remote Operation Services) protocols. It will document the design of the system and the code written to implement those protocols.

The scope of this document is the protocols of Efficient Short Remote Operation Services.

Who to contact

For more information about this document or its contents, please contact:


```
Neda Communications, Inc.
Phone: +1 425644-8026
Fax:    1+1 425562-9591
E-Mail: info@neda.com
```

Chapter 1

Introduction

1.1 About This Product

Neda's Implementation of ESRO – Source documents ESRO-FULL-SRC, the portable source code of Neda's Implementation of the Efficient Short Remote Operation Services Protocol. This implementation conforms to "RFC-2188" [1].

ESRO-FULL-SRC consists of the Efficient Short Remote Operations Base (ESROS-BASE-SRC) and Efficient Short Remote Operations Test Tools (ESROS-TEST-SRC). ESRO-FULL-SRC relies on Neda's Open C Platform [2]. Open C Platform and related documentation is supplied separately.

ESRO-BASE-SRC is a relatively complete implementation of the ESROS protocol specification. Complete sources for both invoker and performer sides are implemented. The specific features and options implemented are enumerated in Neda's Protocol Implementation Conformance Statement (PICS). Neda's PICS is included as a part of this documentation in Appendix E.

ESRO-FULL-SRC is provided as portable software which may be ported to a variety of environments. Although a great deal of portation specific code and documentation is included, it is important to note that they are provided only as examples. It is only the portable software which is fully supported and maintained.

This implementation can be configured to be utilized in a complex Message Center, handling extremely high volumes, or in a very small, embedded device whose size and efficiency are of the highest importance.

1.2 About This Document

This document supplies the documentation for the following products:

- ESRO-FULL-SRC
- ESRO-BASE-SRC
- ESRO-TEST-SRC

Some of the documentation may not be pertinent to the reader, but it is included for the sake of completeness.

It is recommended that for this document to be of the most use to the reader, they should be extremely familiar with "RFC-2188" [1] and the Open C Platform [2]. There has been no attempt in this document to re-explain anything previously covered in the Protocol Specifications.

For ease of reference, all citations from Neda source code files are printed in boldface type.

Chapter 1 consists of an introduction to the product, ESRO-FULL-SRC, and the whole document.

Chapter 2 describes the Efficient Short Remote Operation Protocol Engine.

Chapter 3 provides information about the interface to ESROS services.

Chapter 4 describes the design and operation of real-world ESROS programs which have been ported to both Unix and MS-DOS platforms. However, it is important to note that they are provided only as examples. It is only the portable software which is fully supported and maintained.

Chapter 5 describes the testing methodologies and tools used to test an ESROS implementations.

Appendices include a Bibliography, a list of relevant Acronyms, ESROS API Example Usage, ESROS Program Man Pages, Trace Bit Definitions and Neda's Protocol Implementation Conformance Statement (PICS).

1.3 Architecture

Figure 1.1, Implementation Architecture depicts the architecture Neda's implementation of the complete ESRO protocols.

ESROS-Daemon is responsible for implementation of ESRO-Protocol ("RFC-2188".[1]) on both invoker and performer sides. ESROS-Daemon exposes the ESROS API (see chapter entitled ESROS API) to its users, the ESROS-User-Daemon and the ESROS-System-Daemon.

1.4 Overview of Software

The software consists of two files:

- ocpForesros-svr4.tar
- esros-full-src.tar

esros-full-src.tar includes:

- Generation scripts to compile and build base.src and test.src directories.
- Portable sources for ESROS protocol engine
- Unsupported system specific portations of ESROS protocol engine.
- Portable sources for ESROS test tools
- Unsupported system specific portations of ESROS test tools.

1.4.1 Getting the sources

From the installation home directory, type tar -xvf ocpForEsros-svr4.tar, to install the Open C Platform. This is required for ESRO-FULL-SRC, ESRO-BASE-SRC and ESRO-TEST-SRC.

By typing tar -xvf esros-full-src.tar, the ESROS user,provider and test programs will be installed.

A brief description of the directory hierarchy follows:

base.src: This directory contain the sources and header files for the ESROS protocol engine

unsupp.svr4: Unsupported System V Rel. 4 specific portation of the protocol engine.

bin: Scripts and utilities for compilation of the software.

include: System wide header files such as "target.h" which define portation specific parameters of the build.

ocp.svr4: Place holder for OCP header files. This directory should be replaced by the relevant complete OCP directory.

results.svr4: Place holder where the results of compilation (e.g., libraries and executables) will be stored.

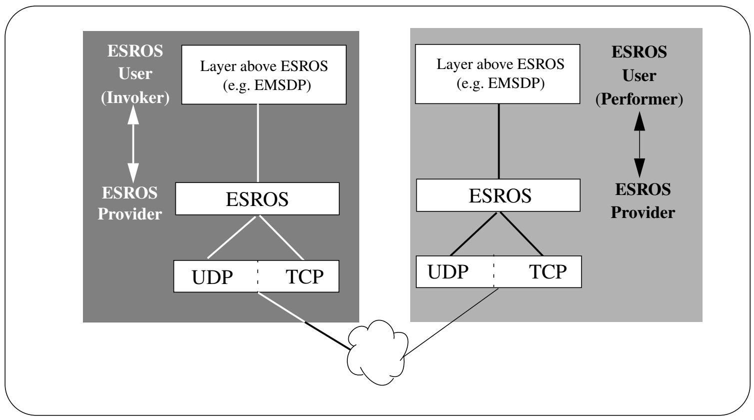

Figure 1.1: Implementation Architecture

test.src: This directory contain the sources and header files for the ESROS test tools.

unsupp.svr4: Unsupported System V Rel. 4 specific portation of the test tools.

It is very important to note that all sub directories under ”unsupp.*” are not considered part of Neda ESROS products and are supplied as example portations only. Eventhough, many of the example programs are described in some detail in this document, they should not be considered a proper part of ESROS products.

1.4.2 Compiling and Building the Software

To build this distribution:

```
source sourceme.csh
```

And then:

```
./buildAll.sh
```

If you don’t have gmake, you may have to edit the build scripts.
To run esros example programs refer to the relevant sections of this document.

Chapter 2

ESRO Protocol Engine

2.1 Introduction

The ESRO layer operates above the Transport layer and provides services to other application layer entities. It provides remote operation services between application layer entities by relieving them of concern over how the remote operation is achieved.

The Neda ESRO Protocol Engine (ESROP) is the Neda implementation of the ESRO protocol. The ESRO protocol is defined in ”RFC-2188” [1].

The Neda ESROP implementation is designed to be independent of the environment to which it will be ported. ESROP software makes minimal assumptions about its target environment.

While this chapter describes how protocol options are implemented and used, it does not attempt to explain the protocol. ESROP interface function calls are modeled after the ESROS service definition,”RFC-2188” [1]. References to concepts defined by the ESRO Protocol Specifications are made throughout the book. We recommend reading the ESRO Protocol Specification documents [1] in conjunction with this document.

ESROP module uses a set of common facilities. These common facilities are described in Open C Platform [1]. We suggest a review of that manual before starting this manual.

The chapter is divided into three basic parts. An overview and concepts of ESROP's functionality can be found in Section2.2. ESROS Service Primitives are described in Section2.3. Section 2.4 and Section2.5 define the upper and lower interface functions, respectively, of ESROP. The environment dependent facilities and network management are discussed in Section2.6 and Section2.7. Portation issues and design of ESROP are discussed in Section2.8 and Section2.9.

2.2 Overview and Concepts

2.2.1 ESROP Service Features

To an ESROP user the ESROP Services offer the means to perform an Operation with another ESROP-User for the purpose of exchanging ESROS Data Units (DU). More than one Operation may be in progress simultaneously between the same pair of ESROP-Users.

2.2.2 Software Architecture

ESROP is designed to be independent of the environment to which it will be ported.

ESROP appears to its user as a link module with four defined interfaces. The ESROP module's interfaces are shown Figure 2.1.

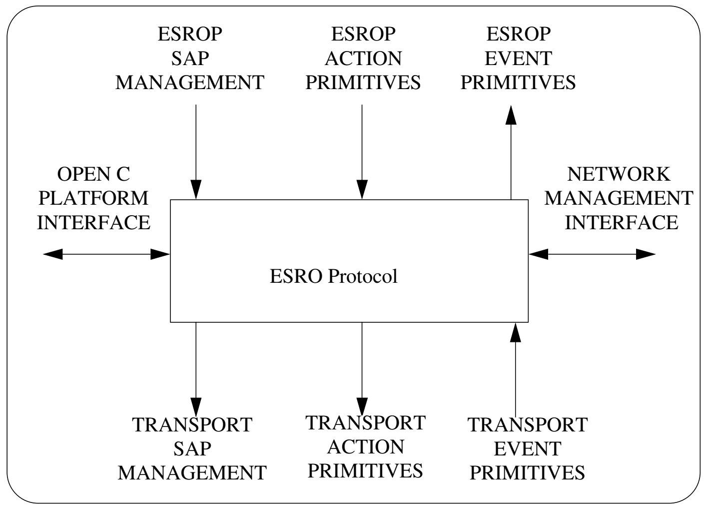

Figure 2.1: Neda's ESROP Module and its Interfaces

2.2.2.1 Upper Interface

The ESROP upper interface is a series of function calls (primitives). Each function call accepts a group of arguments (parameters).

The Neda ESROP upper interface provides the ESROP-User with primitives which match the definitions of ESROS services [1]. Each function call is non-blocking and asynchronous.

ESROP requests and responses, collectively referred to as ESROP action primitives, are function calls to the ESROP software module. ESROP action primitives are invoked by the ESROP-User. The code for ESROP action primitives is provided by the ESROP software module.

ESROP indications and confirmations, collectively referred to as ESROP event primitives, are function calls to the ESROP-User module. ESROP event primitives are invoked by the ESROP software module. The code for ESROP event primitives is expected to be provided by the ESROP-User.

Section2.4 describes in detail the ESROP upper interface primitives and their parameters.

2.2.2.2 Lower Interface

The lower interface of the ESROP software module matches the upper interface of the Transport software module.

The ESROP lower interface is a series of function calls (primitives). Each function call accepts a group of arguments (parameters).

Neda ESROP lower interface primitives match the ESRO Service definitions [2]. Lower interface function calls are non-blocking and asynchronous.

Transport requests and responses, collectively referred to as Transport action primitives, are function calls to the Transport software module. Transport action primitives are invoked by the ESROP software module. The code for Transport action primitives is expected to be in the Transport software module.

Transport indications and confirmations, collectively referred to as Transport event primitives, are function calls to the ESROP software module. Transport event primitives are invoked by the Transport software module. The code for Transport event primitives is provided in the ESROP software module.

2.2.2.3 Timer and Data Unit Management Interface

ESROP relies on the availability of a group of facilities for data unit and timer manipulation. A common high-level interface for these facilities is used by the ESROP module. This interface is defined in Open C Platform document, [2] and reviewed later.

2.2.2.4 Network Management Interface

A set of counters is locally maintained by the ESROP software module. These counters, which may be used for Network Management, are described later.

2.3 ESRO Service Primitives

This section describes the service primitives provided by the ESROP module, and the constraints on the sequence in which the ESROP primitives may occur. Each ESROP-User interacts with the ESROP module through one or more ESROP-SAPs.

The following is a list of ESRO service primitive names:

The Neda ESROP upper interface conforms to the ESRO Service Definition [2]. The constraints on the sequence in which ESROP primitives may occur are explained in Reference [2].

ESRO Service Primitives
ESROS-INVOKE.request ESROS-INOVKE-P.confirm ESROS-INVOKE.indication
ESROS-RESULT.request ESROS-RESULT.indication ESROS-RESULT.confirm
ESROS-ERROR.request ESROS-ERROR.indication ESROS-ERROR.confirm
ESROS-FAILURE.indication

Table 2.1: ESRO Service Primitives

Function	Description
ESROP_sapBind	Bind an ESROP-SAP and register an ESROP-User.
ESROP_sapUnbind	Unbind an ESROP-SAP and deregister an ESROP-User.

Table 2.2: ESROS-SAP Management

2.3.1 SAP Management

An ESROP-User must create an ESROP-SAP before it can use any of the services provided by the ESROP module. Creation of an ESROP-SAP is accomplished through the ESROP_sapBind function. Parameters to ESROP_sapBind communicate to the ESROP module both an ESRO-SAP selector address and a set of functions for handling event primitives for that ESROP-SAP. ESROP event primitives are:

- ESROS-INVOKE.indication
- ESROS-RESULT.indication
- ESROS-ERROR.indication
- ESROS-FAILURE.indication

Deletion of an ESROP-SAP is accomplished through the ESROP_sapUnbind function. A summary of Neda ESROP-SAP management facilities follows.

2.3.2 Operation Invocation

The sequence of ESROP primitives in an OPERATION is illustrated in the time sequence diagram below.

To initiate an ESROP operation, the invoker ESROP-User entity issues an ESROS-INVOKE.request at the ESROP layer interface by invoking the function ESROP_invokeReq. The performer ESROP entity's ESROP-SAP is specified as one of the parameters of this action primitive.

The ESROS-INVOKE-P.confirm primitive is communicated to the invoker user through the return value/parameter of the ESROP_invokeReq function.

An ESROS-INVOKE.indication event primitive is generated at the performer ESROP entity's ESROP-SAP through the invocation of the (*ESROP_invokeInd)() function associated with the performer ESROP-SAP.

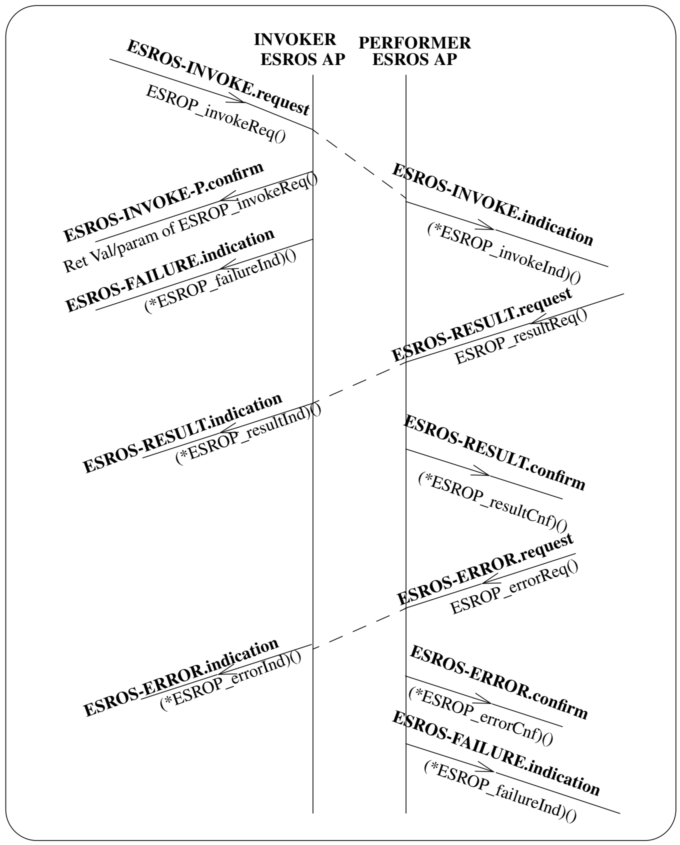

Figure 2.2: Time sequence diagram for ESRO Services

Service Primitive Name	Neda Function Name	Source
ESROESROS-INVOKE.request	ESROP_invokeReq()	Invoker user
ESROS-INVOKE-P.confirm	Ret Val of ESROP_invokeReq()	Provider
ESROS-INVOKE.indication	(*ESROP_invokeInd)()	Provider
ESROS-RESULT.request	ESROP_resultReq()	Performer user
ESROS-RESULT.indication	(*ESROP_resultInd)()	Provider
ESROS-RESULT.confirm	(*ESROP_resultCnf)()	Provider
ESROS-ERROR.request	ESROP_errorReq()	Performer user
ESROS-ERROR.indication	(*ESROP_errorInd)()	Provider
ESROS-ERROR.confirm	(*ESROP_errorCnf)()	Provider
ESROS-FAILURE.indication	(*ESROP_failureInd)()	Provider

Table 2.3: Service Primitives and corresponding functions

The performer ESROP-User can accept the operation and communicate the results by generating an ESROS-RESULT.request at the ESROP layer interface by invoking the function ESROP_resultReq. The performer ESROP-User can issue an ESROS-ERROR.request by invoking the function ESROP_errorReq.

An ESROS-RESULT.confirm or ESROS-ERROR.confirm event primitive is generated at the performer ESROP entity ESROP-SAP through the invocation of the (*ESROP_resultCnf)() or (*ESROP_errorCnf)() function associated with the performer ESROP-SAP.

An ESROS-RESULT.indication or ESROS-ERROR.indication event primitive is generated at the invoker ESROP entity ESROP-SAP through the invocation of the (*ESROP_resultInd)() or (*ESROP_errorInd)() function associated with the invoker ESROP-SAP.

A summary of all operation primitives appears below in Table 2.3:

The OPERATION may fail due to either the inability of the ESROS provider to transmit the INVOKE PDU or the unwillingness of the ESROS performer user to accept an ESROS-INVOKE.indication. These cases are described later in this chapter. The OPERATION may also fail as a result of the failure in delivery of RESULT or ERROR PDU. In such cases an ESROS-FAILURE.indication event primitive is issued at the invoker or performer ESROP-SAP through the invocation of the (*ESROP_failureInd)() function.

2.4 Upper Interface Functions

2.4.1 SAP Management

The ESROP-User calls ESROP SAP management functions to register or deregister as an ESROS service user.

2.4.1.1 Bind an ESROP-User

```
Int
ESROP_sapBind   (ESROP_SapDescsapDesc,
                          ESROP_SapSelsapSel,
                          ESROP_FunctionalUnit functionalUnit,
                          int (*invokeInd)(ESROP_SapSel,
                                ESROP_SapSel,
                                T_SapSel */
                                N_SapAddr */
                                ESROP_InvokeDesc,
                                ESROP_OperationValue,
                                ESROP_EncodingType,
```

```
                DU_View),
         int (*resultInd) (ESROP_InvokeDesc,
                ESROP_UserInvokeRef,
                ESROP_EncodingType,
                DU_View),
         int (*errorInd)  (ESROP_InvokeDesc,
                ESROP_UserInvokeRef,
                ESROP_EncodingType,
                ESROP_ErrorValue,
                DU_View),
         int (*resultCnf) (ESROP_InvokeDesc,
                ESROP_UserInvokeRef),
         int (*errorCnf)( ESROP_InvokeDesc,
                ESROP_UserInvokeRef),
         int (*failureInd)(ESROP_InvokeDesc,
                ESROP_UserInvokeRef,
                ESROP_FailureValue))
```

This function binds an ESROP-User at the ESROP layer by creating an ESROP-SAP.

The sapSel argument specifies the ESROP-SAP Selector Address to be associated with this service user. The ESROP-SAP descriptor value is returned to the caller through sapDesc argument.

The functionalUnit argument specifies the method of handshaking used by the SAP.

The remaining arguments specify the addresses of the callback functions (and their arguments) that the SAP should invoke upon the occurrence of specific events. For instance, invokeInd points to the function that is executed when a performer ESROP receives an invocation.

ESROP_sapBind returns 0 on successful completion. It returns a negative value if unsuccessful.

2.4.1.2 Unbind an ESROP-User

```
SuccFail
ESROP_sapUnbind(ESROP_SapSel sapSel)
```

This function unbinds an ESROP-User. If there are any remaining invocations associated with this SAP for that ESROP-User, the pending or in process requests are dropped and if there are any events queued up for this ESROP-User's SAP, they are discarded.

The sapSel argument specifies the ESROP-SAP currently in use by the caller.

This function returns 0 on successful completion. It returns a negative value if unsuccessful.

2.4.2 ESROP Action Primitives

2.4.2.1 ESROS-INVOKE.request

```
Int
ESROP_invokeReq(ESROP_InvokeDescinvokeDesc,
                ESROP_UserInvokeRefuserInvokeRef,
                ESROP_SapSellocESRSap,
                ESROP_SapSelremESRSap,
                T_SapSel*remTsap,
                N_SapAddr*remNsap,
                ESROP_OperationValueopValue,
                ESROP_EncodingTypeencodingType,
                DU_View   parameter)
```

The invoker ESROP-User invokes this function to initiate the invocation of an ESROS operation. The ESROP-User specifies both local and remote ESROP-SAP. The locESROSap argument contains the ESROP-SAP Selector of the invoker, while the remESROSap argument contains the ESROP-SAP of the performer. The remTsap specifies the Transport SAP Selector of the performer and remNsap specifies the Network SAP of the performer. Invoker ESROP-User passes an invocation identifier (userInvokeRef) to the ESROP which is returned to it in all future references to that specific operation.

opValue is the operation value. The invoker ESROP-User can specify its desired encoding type through the encodingType argument. parameter is the operation's parameter.

An invocation descriptor is passed to the caller of the function (invoker ESROP-User) through the invokeDesc argument to be used to identify the ESROS invocation in future references. If ESROP fails in initializing the invocation operation a negative value is returned by the function, otherwise 0 is returned on successful completion of the function call.

2.4.2.2 ESROS-RESULT.request

```
Int
ESROP_resultReq(ESROP_InvokeDescinvokeDesc,
ESROP_UserInvokeRefuserInvokeRef,
ESROP_EncodingTypeencodingType,
DU_View parameter)
```

When ESROP performer user accepts an invocation indication, it invokes this function to request transmission of the result of the operation.

invokeDesc is the invocation identifier that was previously obtained through the ESROS-INVOKE.indication primitive on performer-user side. Performer ESROP-User passes the userInvokeRef to ESROP which is returned to it in all future references to the operation. encodingType reflects the encoding type of the parameter. parameter is the operation's parameter.

This function returns 0 on successful completion. If the function is unsuccessful, it returns a negative value.

2.4.2.3 ESROS-ERROR.request

```
Int
ESROP_errorReq(ESROP_InvokeDesc invokeDesc,
                              ESROP_UserInvokeRefuserInvokeRef,
                              ESROP_EncodingType encodingType,
                              ESROP_ErrorValue errorValue,
                              DU_View parameter)
```

When ESROP performer user accepts an invocation indication, it invokes this function to request transmission of the error response to the invoker of the operation.

invokeDesc is the invocation identifier that was previously obtained through the ESROS-INVOKE.indication primitive on performer-user side. encodingType reflects the encoding type of the parameter. parameter is the operation's parameter.

This function returns 0 on successful completion. If the function is unsuccessful, it returns a negative value.

2.4.3 ESROS Event Primitives

2.4.3.1 ESROS-INVOKE.indication

```
Int
(*ESROP_invokeInd)(ESROP_SapSellocESROSap,
                                  ESROP_SapSelremESROSap,
```

```
                                    T_SapSel **remTsap,
                                    N_SapAddr **remNsap,
                                    ESROP_InvokeDescinvokeDesc,
                                    ESROP_OperationValueoperationValue,
                                    ESROP_EncodingTypeencodingType,
                                    DU_Viewparameter)
```

The performer ESROP invokes this function to communicate an invoke indication to the ESROP-User. ESROP specifies both the invoker ESROP-SAP Selector and the performer ESROP-SAP Selector.

The remTsap argument contains the Transport SAP Selector of the performer and remNsap specifies the Network SAP of the performer. The encodingType argument specifies the encoding type of the parameter. parameter is the operation's parameter.

This function returns 0 in the case of success and a negative value if unsuccessful.

2.4.3.2 ESROS-RESULT.indication

```
Int
(*ESROP_resultInd)(ESROP_InvokeDescinvokeDesc,
                                    ESROP_UserInvokeRefuserInvokeRef,
                                    ESROP_EncodingTypeencodingType,
                                    DU_View        parameter)
```

The ESROP invokes this function to communicate to the ESROP-User a result indication.

invokeDesc is the invocation identifier that was previously communicated to the invoker user through ESROS-INVOKE-P.confirm. userInvokeRef is the user's invocation identifier that was passed to ESROP by invoker-user at the time of initiation of ESROS-INVOKE.request. encodingType reflects the encoding type of the parameter used by the performer ESROP-User. parameter is the operation's parameter.

This function returns 0 in the case of success and a negative value if unsuccessful.

2.4.3.3 ESROS-ERROR.indication

```
Int
(*ESROP_errorInd) (ESROP_InvokeDescinvokeDesc,
                                    ESROP_UserInvokeRefuserInvokeRef,
                                    ESROP_EncodingTypeencodingType,
                                    ESROP_ErrorValueerrorValue,
                                    DU_Viewparameter)
```

The ESROP invokes this function to communicate to the invoker ESROP-User an error response to its invocation.

invokeDesc is the invocation identifier that was previously communicated to the invoker user through ESROS-INVOKE-P.confirm (ESROP_invokeReq parameter). userInvokeRef is the user's invocation identifier that was passed to ESROP by invoker-user at the time of initiation of ESROS-INVOKE.request. encodingType reflects the encoding type of the parameter used by the performer ESROP-User. parameter is the operation's parameter.

This function returns 0 in the case of success and a negative value if unsuccessful.

2.4.3.4 ESROS-FAILURE.indication

```
Int
(*ESROP_failureInd) (ESROP_InvokeDescinvokeDesc,
                                    ESROP_UserInvokeRefuserInvokeRef,
                                    ESROP_FailureValuefailureValue)
```

The ESROP invokes this function to communicate to the ESROP-User a failure indication.

invokeDesc is the invocation identifier that was previously communicated to the invoker ESROP-User through ESROS-INVOKE-P.confirm (ESROP_invokeReq parameter) or communicated to the performer ESROP-User through ESROS-INVOKE.indication. userInvokeRef is the user's invocation identifier that was passed to ESROP by invoker-user at the time of initiation of ESROS-INVOKE.request (ESROP_invokeReq) on invoker side or ESROS-RESULT/ERROR.request on performer side.

The failureValue argument specifies the failure reason.

This function returns 0 in the case of success and a negative error value if unsuccessful.

2.4.3.5 ESROS-RESULT.confirm

```
Int
(*ESROP_resultCnf) (ESROP_InvokeDescinvokeDesc,
                                ESROP_UserInvokeRefuserInvokeRef)
```

The ESROP invokes this function to communicate the confirmation of the result.

invokeDesc is the invocation identifier that was previously communicated to the performer ESROP-user through ESROS-INVOKE.indication. userInvokeRef is the user's invocation identifier that was passed to ESROP by performer ESROP-User at the time of initiation of ESROS-RESULT.request (ESROP_resultReq).

This function returns 0 in the case of success and a negative value if unsuccessful.

2.4.3.6 ESROS-ERROR.confirm

```
Int
(*ESROP_errorCnf)(ESROP_InvokeDescinvokeDesc,
                              ESROP_UserInvokeRefuserInvokeRef)
```

The ESROP invokes this function to communicate the confirmation of the error.

invokeDesc is the invocation identifier that was previously communicated to the performer ESROP-user through ESROS-INVOKE.indication. userInvokeRef is the user's invocation identifier that was passed to ESROP by performer ESROP-User at the time of initiation of ESROS-ERROR.request (ESROP_errorReq).

This function returns 0 in the case of success and a negative value if unsuccessful.

2.5 Lower Interface Functions

ESROP relies on the connectionless or connection oriented services provided by Transport layer based on the SDU size. The following sections describe the Transport services assumed by ESROP.

2.5.1 Connectionless interface

2.5.1.1 UDP SAP Bind

```
UDP_SapDesc
UDP_sapBind (T_SapSelsapSel,
            int (*dataInd)(T_SapSel,
                  N_SapAddr,
                  T_SapSel,
                  N_SapAddr,
                  DU_View))
```

This function binds the ESROP to the lower layer by creating a UDP-SAP.

The sapSel argument specifies the local T-SAP Selector Address to be associated with this UDP-SAP.

The dataInd argument specifies the callback function that gets executed when the lower layer has data for the ESROP layer.

The UDP-SAP descriptor value is returned to the caller on successful completion. Otherwise a NULL value is returned.

2.5.1.2 UDP SAP Unbind

```
SuccFail
UDP_sapUnBind (T_SapSel *sapSel)
```

This function unbinds an ESROP from the lower layer. If there are any remaining requests associated with this SAP, the pending or in process requests are dropped and if there are any events queued up for this, they are dumped.

The sapSel argument specifies the UDP-SAP currently in use by the caller.

This function returns SUCCESS on successful completion and FAIL if unsuccessful.

2.5.2 Action Primitives

```
SuccFail
UDP_dataReq (UDP_SapDesc locSapDesc,
T_SapSel      *remTsapSel,
N_SapAddr     *remNsapAddr,
DU_View       udpSdu)
```

This function is called by the ESROP layer when it needs to send a TSDU.

The locSapDesc argument identifies the local UDP SAP in use. remTsapSel specifies the destination SAP address. remNsapAddr specifies the Network SAP address of the remote peer. udpSdu is a buffer containing the TSDU.

2.5.3 Event Primitives

```
Int
(*dataInd) (T_SapSel *remTsapSel,
N_SapAddr *remNsapAddr,
T_SapSel *locTsapSel,
N_SapAddr *locNsapAddr,
DU_View udpSdu))
```

The UDP layer invokes this function to communicate received SDU's to the UDP user.

remTsapSel specifies the transport SAP address of the remote peer. remNsapAddr specifies the Network SAP address of the remote peer. udpSdu is a buffer containing the TSDU.

Note: locTsapSel and locNsapAddr are reserved for future use. Do not rely on their values.

2.6 Environment Interface

This section describes the environment dependent facilities used by the ESROP module. The Neda publication, Open C Platform [2], describes these facilities in detail.

2.6.1 Data Unit Management

ESROP uses all of the data unit management facilities described in Open C Platform [1].

No	Counter name	Contents
1	esrop_pduRetranCounter	number of PDU Retransmissions
2	esrop_completeOperationCounter	number of Completed Operations
3	esrop_protocolErrorsCounterd	number of Protocol Errors
4	esrop_pduSentCounter	number of PDUs Sent
5	esrop_invokePduRetranCounter	no of Invoke PDU Retransmissions
6	esrop_badAddrCounter	number of Bad Addresses
7	esrop_opRefusedCounter	number of Operations Refused
8	esrop_udpSduSentCounter	number of UDP SDU's Sent
9	udpSdu_rcvd	number of UDP SDU's received
10	udp_pdu_bad	number of bad UPD PDU's

Table 2.4: Network Management Counters

2.6.2 Timer Management

ESROP uses all of the timer management facilities described in Open C Platform [1].

2.7 Network Management

Several counters are locally maintained to be used for Network Management in each ESROP entity (see Table 2.4).

2.8 Portation Notes

This section describes some of the more important points which should be considered when porting ESROP to a target environment.

2.8.1 Unresolved Symbols

If all ESROP layer modules are linked together, the symbols listed in Table 2.5 are unresolved for the ESROP module.

The remaining symbols are associated with the Open C Platform[1] which contains detailed information about each of these facilities.

2.8.2 Configuration

Several compile time constants must be configured before the ESROP module can be used. These constants are either defined by ESROP or ESROP User. Some of these configuration parameters can be statically changed at run time by changing the ESROP configuration file. The following table is the list of these configuration parameters.

2.8.3 Tracing

To facilitate the portation process of ESROP, some tracing capabilities have been built into this product.

To enable the tracing feature, the compile time constant TM_ENABLED must be defined.

Tracing level can be controlled through definition of trace masks in the ESROP configuration file.

2.8.4 Differences Between Unix and MS-DOS/Windows Portations

The major differences between the Unix and MS-DOS/Windows portations are isolated to several discrete locations in the code. These differences are discussed in the following sections.

Symbol	Module
DU_alloc	Data Unit Module
DU_free	Data Unit Module
DU_link	Data Unit Module
G_duMainPool	Global Module (integration of all modules)
INET_in_addrToNsapAddr	Internet Module
INET_nsapAddrToin_addr	Internet Module
INET_portNuToTsapSel	Internet Module
INET_tsapSelToPortNu	Internet Module
NM_incCounter	Network Management Module
N_sapAddrCmp	Network Service Access Point Module
PF_crc16	Public Facilities Module
QU_init	Queue Management Module
QU_insert	Queue Management Module
QU_move	Queue Management Module
QU_remove	Queue Management Module
SAP_selCmp	Service Access Point Module
SEQ_elemObtain	Sequence Module (mem allocation for lists)
SEQ_elemRelease	Sequence Module
SEQ_poolCreate	Sequence Module
TMR_cancel	Timer Management Module
TMR_create	Timer Management Module
TMR_getData	Timer Management Module
TMR_getDesc	Timer Management Module
TM_hexDump	Trace Module
TM_open	Trace Module
TM_prAddr	Trace Module
tm_loc (TM_loc)	Trace Module
tm_here (TM_here)	Trace Module
tm_trace (TM_trace)	Trace Module

Table 2.5: Unresolved Symbols of ESROP

Symbol	Definition
ESROP_K_UdpSapSel	UDP SAP selector (currently 2002)
ESROP_SAPS	Total number of SAP entries.
ESROP_INVOEKS	Total number of concurrent invocations
INVOKE_PDU_SIZE	Invoke PDU size
NUMBER_OF_TIMERS	Maximum number of timers
MAX_SAPS	Maximum number of active SAPs
MAXBFSZ	Buffer size of Data Unit module
BUFFERS	Total number of DU buffers
VIEWS	Total number of DU views

Table 2.6: ESROP Configuration Parameters

2.8.4.1 Multiple Processes vs. Single Process

The most significant difference between the two portations lies in the manner in which an ESROS user communicates with ESROS itself. Under Unix, ESROS runs as a separate task. There may thus be one or more ESROS users that invoke ESROS primitives via an interprocess communication mechanism provided by the UPQ_BSD_ module.

MS-DOS, of course, does not support such a mechanism. Therefor, ESROS and ESROS users are linked together as single executable. This is the same for Windows. As explained in the previous sections, ESROP has a call-back interface to the upper layer. However, there are two different API's available to the ESROS user (see Section, ESROS API). In the case of applications that use the Call-back API, the inter-process communication module is eliminated for one-process model. In the case of Function Call API, in place of the interprocess communication mechanism, there is a a module that emulates the UPQ_BSD_ facilities. This module, named UPQ_SIMU_, uses disk files to simulate the interprocess communication mechanism.

The developer of an ESROS application who has to port between these two environments will be chiefly concerned with the ESROS user's make file. Under Unix and for two process model, the ESROS user application is linked to the UPQ_BSD_ library. Under MS-DOS it is linked to the UPQ_SIMU_ library. In addition, the MS-DOS user must link to the libraries containing the ESROS code. The following excerpts illustrate this principle.

Libraries used building Unix ESROS user application as a separate process

```
USER_SH = $(LIBS_PATH)/esro_ushcb.a
UPQ = $(LIBS_PATH)/upq_bsd.a
GF = $(LIBS_PATH)/gf.a
```

Libraries used building MS-DOS ESROS user application in one process with Function Call API

```
USER_SH = $(LIBS_PATH)\esro_ush.lib
PRVDR_SH = $(LIBS_PATH)\sp_shell.lib
UPQ_SIMU = $(LIBS_PATH)\upq_simu.lib
UDP_IF = $(LIBS_PATH)\udp_pco.lib
ESROP_SH = $(LIBS_PATH)\esrop_sh.lib
PROT_ENG = $(LIBS_PATH)\esroprot.lib
GF = $(LIBS_PATH)\gf.lib
```

2.8.4.2 Libraries used building MS-DOS ESROS user application in one process with Call-back API

```
UDP_IF = $(LIBS_PATH)\udp_pco.lib
ESROP_SH = $(LIBS_PATH)\esrop_sh.lib
PROT_ENG = $(LIBS_PATH)\esroprot.lib
GF = $(LIBS_PATH)\gf.lib
SF = $(LIBS_PATH)\sf.lib
FSM = $(LIBS_PATH)\fsm.lib
```

2.8.4.3 Scheduler Module (SCH_)

SCH_ module can be used for scheduling the program's modules.

One of the common usages of SCH_ module is scheduling of further processing within the same module. This happens most often to prevent re-entry to non-re-entrant code. For more information about the Scheduler module refer to OPEN C Platform document.

2.8.4.4 Timer Module (TMR_)

The TMR_ module defines a model and an interface for providing timer facilities to Open C Layers, regardless of the environment, provided that all implementations of the TMR_ module conform to the interface defined here. For more information about the Timer module refer to OPEN C Platform document.

2.8.4.5 FLEX & BISON

FLEX is a DOS portation of the lex utility commonly found on Unix systems. BISON is a DOS portation of the yacc utility. These utilities are used in the compilation of the ESROS Scenario interpreter, ESROSSI.

2.9 Design Overview

An overview of the structure of the ESROP module appears in Figure 2.3.

The ESROS service user interface consists of the USER-IN Interface, USER-OUT Interface, and the SAP Map.

The USER-IN Interface is the module that deals with ESROP action primitives. userin.c contains the source to this module.

The USER-OUT Interface is the module that deals with ESROP event primitives. userout.c contains the source to this module.

The SAP Map deals with multiplexing of ESROP-Users. esrop_sap.c contains the functions related to this module.

The ESROP Finite State Machine is the heart of ESROP. invokact.c/invact2.c, perfact.c/perfact2.c, clinvktd.c/clinvtd2.c, and clperftd.c/clpertd2.c contain the functions related to this module.

The PDU Formatter deals with encoding of information into PDUs. The pduout.c contains the source to this module.

The PDU Parser deals with decoding of information from PDUs. The pduin.c contains the source to this module.

2.9.1 SAP Management

ESROP has the capability of supporting communication with several entities in the layer above. This is accomplished through the concept of an ESROP-SAP. struct SapInfo is the basic structure used. The SapInfo structure associates a series of event primitives with an ESROP_SapSel.

SapInfo structures are maintained as doubly linked lists. SapInfoSeq contains pointers to the first and last SapInfo structures. ESROP_init creates a pool of ESROP-SAPS SapInfo. ESROP_SAPS is a configuration parameter that specifies the maximum number of ESRO-SAPs supported by ESROP.

Each SapInfo structure maintains a list of active invocations associated with it. The InvokeInfoSeq field of SapInfo structure is the head of this list.

2.9.2 ESROP Finite State Machine

The ESROP Finite State Machine (FSM) is the heart of ESROP. Figure 2.4 ESROP Finite State Machine is a high level diagram of ESROP state machine.

Depending on the size of the SDU, the connection oriented or connectionless transition diagram is selected.

2.9.2.1 Events

The inputs to the FSM are:

```
/* User Action Primitives */
#define lsfsm_EvtInvokeReq2
#define lsfsm_EvtErrorReq3
#define lsfsm_EvtResultReq4
```

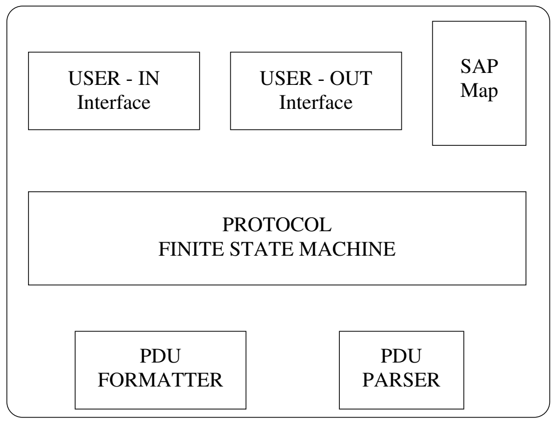

Figure 2.3: ESROP Modules

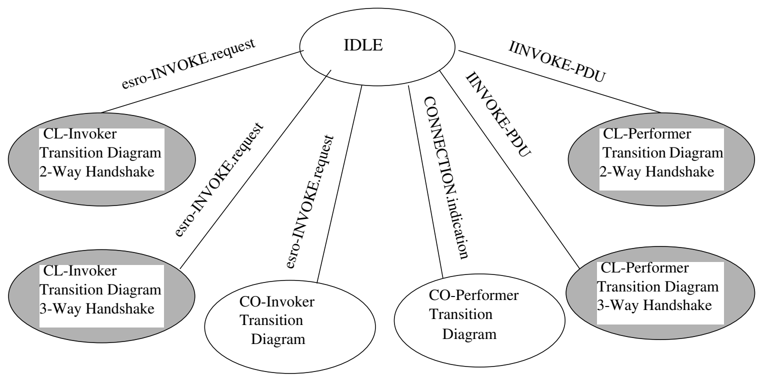

Figure 2.4: ESROP Finite State Machine

```
#define lsfsm_EvtPduInputInd5
    /* In PDU Indications */
#define    lsfsm_EvtPduInvoke6
#define    lsfsm_EvtPduResult7
#define    lsfsm_EvtPduAck8
#define    lsfsm_EvtPduFailure9

#define    lsfsm_EvtTimerInd10
    /* Timer Indications */
#define    lsfsm_EvtInvokePduRetranTimer11
#define    lsfsm_EvtResultPduRetranTimer12
#define    lsfsm_EvtRefNuTimer13
#define    lsfsm_EvtInactivityTimer14
#define    lsfsm_EvtLastTimer15
#define    lsfsm_EvtPerfNoResponseTimer16
```

Each of these events have some information associated with them:

```
typedef union FSM_EventInfo {

    struct InvokeReq {
ESRO_SapSel     remESROSap;
T_SapSel        *remTsap;
N_SapAddr       *remNsap;
ESROP_OperationValue    opValue;
ESROP_EncodingType      encodingType;
DU_View parameter;
} invokeReq;

    struct ResultReq {
Bool    isResultNotError;
ESROP_EncodingType      encodingType;
DU_View parameter;
} resultReq;

    struct ErrorReq {
Bool    isResultNotError;
ESROP_EncodingType      encodingType;
ESROP_ErrorValue        errorValue;
DU_View parameter;
} errorReq;

    struct InternalInfo {
Int     expiredTimerName;
} internalInfo;

    struct TmrInfo {
Int     name;7%)
long    subscript;
Int     datum;)
} tmrInfo;
```

```
} FSM_EventInfo;
```

Each time one of these events is observed by the USER-IN Interface module, by the ESRO-PDU PARSER module, or by the Timer Management Module, the event code and the information associated with it are passed to the event processor (esropfsm).

2.9.2.2 Event Processor

```
PUBLIC Int
FSM_runMachine (FSM_Machine *machine, FSM_EventId evtId)
```

FSM_runMachine contains the information regarding the current state of an ESROS operation. evtId is an event ID corresponding to the event that has been observed.

Each state has an entry and an exit functions associated to it.

```
typedef struct FSM_State {
    Int (*entry)(FSM_Machine *machine,
                 FSM_UserData *userData,
                 FSM_EventId evtId);
    Int (*exit)(FSM_Machine *machine,
                FSM_UserData *userData,
                FSM_EventId evtId);
    struct FSM_Trans *trans;
    String name;
} FSM_State;
```

Using the information regarding the current state and the observed event, FSM_runMachine looks into the link list of transitions and finds the next state corresponding to the observed event. The transitions are a link list of:

```
typedef struct FSM_Trans {
    FSM_EventId evtId;
    Bool (*predicate) (FSM_Machine *machine,
                      FSM_UserData *userData,
                      FSM_EventId evtId);
    Int (*action)(FSM_Machine *machine,
                  FSM_UserData *userData,
                  FSM_EventId evtId);
    FSM_State *nextStatePtr;
    String name;
} FSM_Trans;
```

The exit function of the current state is called. Then the action function is executed. After that, the current sate is changed to the next state and the entry function of the next state is called.

2.9.2.3 State Information

Information regarding the state of each invocation is maintained in a structure FSM_State. FSM_State contains various information about the current state of a connection.

Name	Function	Variable name
retransTimeout	Retransmission timer interval	invokeInfo-¿retransTimeout
refKeepTime	Reference Number expiration time	invokeInfo-¿refKeepTime
rwait	Last timer	invokeInfo-¿rwait
inactivityDelay	Inactivity time	invokeInfo-¿inactivityDelay
perfNoResponse	Performer response time-out	invokeInfo-¿perfNoResponse

Table 2.7: Timers

```
typedef struct FSM_State {
    Int (*entry)(FSM_Machine *machine,
                 FSM_UserData *userData,
                 FSM_EventId evtId);
    Int (*exit)(FSM_Machine *machine,
                FSM_UserData *userData,
                FSM_EventId evtId);
    struct FSM_Trans *trans;
    String name;
} FSM_State;
```

2.9.3 ESROS Timers

The timers of ESROP are statically defined as follows.

2.9.4 ESROP PDU Parser

lower_dataInd [lowerind.c] delivers an TSDU to ESROP.

inpdu reads the PDU header of an incoming PDU. It constructs the PDU structure and strips the header from the data buffer. If there is no data in the buffer, it frees it. At the completion of inpdu, ESRO-PDU contains the information about the received PDU. This information is then used by finite state machine.

A new machine is created through esrop_invokeInit facility.

2.9.5 ESROP PDU Formatter

Formatting of ESRO-PDUs is accomplished through a set of functions contained in pduout.c.

2.9.6 Operation Invocation

The sequence of ESROS primitives in a successful operation is next chapters. This section briefly describes the procedures that ESROP performs to invoke an operation. The description here does not deal with all possible state transitions; only the normal state transitions are described to give a general view to the reader. It is recommended that this section be read in conjunction with the source code.

2.9.6.1 Invoker ESROP-Provider

Upon invocation of ESROS-INVOKE.request [userin.c], the invoker ESROP-Provider allocates a new machine and sets up the parameters relating to the invoke request event. Then delivers the ESROS-INVOKE.request event to the event processor FSM_runMachine. The state machine is assumed to be in the start state. The resulting action transition tr_clInvoker01/tr_2clInvoker01 [invokact.c/invact2.c] saves the remote address and the parameter which the user wishes to invoke. An invoke reference number which will be used to identify the invocation is created. The

ESRO-INVOKE-PDU is formed and transmitted. A retransmission timer is also initialized. The resulting state is ESRO-INVOKE-PDU Sent state.

In ESRO-INVOKE-PDU Sent state, when the retransmission timer expiration event occurs, it indicates that the retransmission interval has expired since the last ESRO-INOVKE-PDU transmission, or the first transmission after ESROS-INVOKE.request. The resulting action transition tr_clInvoker02/tr_2clInvoker02 (invokact.c/invact2.c) retransmits the ESRO-INVOKE-PDU while the number of retransmissions is less than maximum number of retransmissions (invoke-¿nuOfRetrans). The retransmission counter is incremented and when the maximum number of retransmissions is reached, the last timer is started. The retransmission continues to await the arrival of the ESRO-RESULT-PDU or ESRO-ERROR-PDU, or last timer time out (maximum number of retransmissions), or receipt of ESRO-FAILURE-PDU.

In ESRO-INVOKE-PDU Sent state, when the last timer expiration event occurs after maximum retransmission of ESRO-INVOKE-PDU's, the tr_clInvoke03/tr_2clInvoker03 (invokact.c/invact2.c) action function issues an ESROS-FAILURE.indication primitive and initializes the reference number timer. The next state is the Invoker Reference Number Wait state that waits for the reference number timer to expire.

On receipt of ESRO-RESULT-PDU or ESRO-ERROR-PDU in the ESRO-INVOKE-PDU Sent state, the action and next state depends on the service mode and eventually 2-Way or 3-Way handshake. In the case of 3-Way handshake, the tr_clInvoker04 (invokact.c) action function sends an ESRO-ACK-PDU, issues ESROS-RESULT.indication or ESROS-ERROR.indication primitive, and initializes the inactivity timer. The resulting state is ESRO-ACK-PDU-Sent state. In the case of 2-Way handshake, the tr_2clInvoker04 (invact2.c) action function issues ESROS-RESULT.indication or ESROS-ERROR.indication primitive, and initializes the reference number timer.

In ESRO-INVOKE-PDU Sent state, on receipt of an ESRO-FAILURE-PDU, the tr_clInvoker05/tr_2clInvoker05 action (invokact.c/invact2.c) issues an ESROS-FAILURE.indication primitive with User Not Responding failure cause, and initializes the reference number timer. The resulting state is wait state for invoker reference number to expire.

In the case of 3-Way handshake, after receiving the ESRO-RESULT-PDU or ESRO-ERROR-PDU and going to ESRO-ACK-PDU send state, the duplicate ESRO-RESULT-PDU or ESRO-ERROR-PDU causes the action function tr_clInvoker07 (invokact.c) to initialize the inactivity timer and send another ESRO-ACK-PDU. The state machine stays in the ESRO-ACK-PDU Send state until the inactivity timer time-out.

In the Invoker RefNu Wait state, when the reference number timer time-out event occurs, the action tr_clInvoker08/tr_2clInvoker07 (invokact.c/invact2.c) releases the invoke reference number and the state machine goes to idle state.

In Invoker Reference Number Wait state, when ESRO-RESULT-PDU or ESRO-ERROR PDU is received, the action function tr_clInvoker09/tr_2clInvoker06 (invokact.c/invact2.c) resets the invoke reference number timer. The machine doesn't change state.

In the case of 3-Way handshake, In ESRO-ACK-PDU Send state, when the inactivity timer expiration event occurs, the action function tr_clInvoker10 (invokact.c) initializes the reference number timer, and the state is changed to Invoker Reference Number Wait state.

2.9.6.2 Performer ESROP-Provider

When ESROP receives an ESRO-INVOKE-PDU, it parses it, then delivers the INVOKEIND event to the event processor. The state machine is assumed to be in start state. The resulting action transition tr_clPerformer01/tr_2clPerformer01 (perfact.c/perfact2.c) accepts the ESRO-INVOKE-PDU and issues the ESROS-INVOKE.indication event primitive associated with the invoker of the operation. The new state becomes Invoke PDU Received state.

On receipt of an ESROS-RESULT.request primitive from performer user, the action transition tr_clPerformer02/tr_2clPerformer03 (perfact.c/perfact2.c) adds the invoke reference number to the active list, transmits the ESRO-RESULT-PDU or ESRO-ERROR-PDU, and sets the retransmission timer. In the case of 3-Way handshake, the state is changed to ESRO-ACK-PDU Wait state, waiting for ESRO-ACK-PDU. In the case of 2-Way handshake, the Inactivity Timer is set and the state is changed to Result PDU Retransmit.

In the case of 3-Way handshake, on receipt of an ESRO-ACK-PDU in ESRO-ACK-PDU Wait state, the action transition tr_clPerformer03 (perfact.c) initializes the invoke reference number timer and issues an ESROS-RESULT.confirm or an ESROS-ERROR.confirm. The state is changed to Reference Number Wait state.

In the case of 3-Way handshake, when Inactivity Timer expires, the action transition tr_2clPerformer06 (perfact2.c) initializes the invoke reference number timer and issues an ESROS-RESULT.confirm or an ESROS-ERROR.confirm. The state is changed to Reference Number Wait state.

The duplicate ESRO-INVOKE-PDUs in ESRO-INVOKE-PDU received state are ignored, and state machine stays in the same state.

In the case of 3-Way handshake, and in ESRO-ACK-PDU Wait state, when the ESRO-RESULT-PDU or ESRO-ERROR-PDU retransmission timer expires, the action transition tr_clPerformer05 (perfact.c) retransmits the ESRO-RESULT-PDU or ESRO-ERROR-PDU while the number of retransmissions is less than the maximum. The timer for number of retransmissions is incremented. The state machine stays in the ACK-PDU Wait state. The duplicate ESRO-INVOKE-PDUs in ESRO-ACK-PDU Wait state are ignored, and state machine stays in the same state.

In the case of 2-Way handshake, and in Result PDU Retransmit state, when a duplicate Invoke PDU arrives, the action transition tr_2clPerformer05 (perfact2.c) retransmits the ESRO-RESULT-PDU or ESRO-ERROR-PDU and the state machine stays in the same state.

In the ESRO-INVOKE-PDU received state, in the case of any kind of internal failure in ESROP, the action transition tr_clPerformer08/tr_clPerformer04 (perfact.c/perfact2.c) sends an ESRO-FAILURE-PDU. The state is changed to Connectionless Performer Start.

In the case of 2-Way handshake, when Inactivity Timer expires, the action transition tr_clPerformer09 issues an ESROS-FAILURE.indication primitive and initializes the invoke reference number timer. The state is changed to Performer Reference Number Wait state.

In the case of 3-Way handshake and in the ESRO-ACK-PDU Wait state, on expiration of last timer, the action transition tr_clPerformer09 (perfact.c) issues an ESROS-FAILURE.indication primitive and initializes the invoke reference number timer. The state is changed to Performer Reference Number Wait state.

In Performer Reference Number Wait state, the expiration of reference number timer causes the action transition tr_clPerformer10/tr_2clPerformer08 (perfact.c/perfact2.c) to release the invoke reference number. The state is changed to Connectionless Performer Start.

In Performer Reference Number Wait state, on receipt of duplicate ESRO-INVOKE-PDU, the action transition tr_clPerformer7/tr_2clPerformer7 (perfact.c/perfact2.c) resets the invoke reference number timer. The state is not changed.

In the case of 3-Way handshake and in Performer Reference Number Wait state, on receipt of duplicate ESRO-ACK-PDU, the action transition tr_clPerformer11 resets the invoke reference number timer. The state is not changed.

Chapter 3

ESRO API

This chapter provides information about the interface to ESROS services. It is intended for the users of the ESROS sublayer.

The ESROS API is available in two different styles. In the first case the events are made available to the user of the API through function calls. This is known as the Function Call API. Functions of this API implementation all have the ESRO_ prefix. In the second case ESROS events trigger call backs to functions registered by the user of the ESROS API. This is known as Call back API. Functions of this API implementation all have the ESRO_CB_ prefix, in which CB stands for Call Back.

3.1 ESROS With Function Call API

This section provides information about the Function Call API.

The services provided by the ESROS are defined in the ESROS Protocol Specification. The requests and responses are communicated via non-blocking function calls. Remote operation requests, and error and failure indications are communicated to the ESROS user via a call to the ESRO_getEvent function, which may be a blocking call in some implementations.

Remote operation requests, result, error and failure indications are delivered to the ESROS user in an event structure. The reader should consult the following chapters for information about the parameters which make up the structures.

The following subsections describe the ESROS library functions.

3.1.1 Initialize the Parameters

```
PUBLIC ESRO_RetVal
ESRO_init (String configFileName)
```

The argument is defined as follows:

```
configFileName  Config file name
```

configFileName specifies the config file name that contains ESROS initialization values.

3.1.2 Activate ESROS Service Access Point

The ESRO_sapBind function binds an ESRO Service Access Point (ESRO_SAP) to the current user process. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_sapBind(ESRO_SapDesc*sapDesc, /* out */
ESRO_SapSelsapSel
ESRO_FunctionalUnitfunctionalUnit)
```

The arguments are defined as follows:

```
sapDesc      Return value: the SAP descriptor
sapSel       SAP selector
functionalUnitHandshaking   type
```

sapDesc is a pointer to an ESRO_SapDesc structure that is created for the current user.

sapSel identifies the ESROS SAP. If the SAP is in use by another user the function returns an error value.

functionalUnit specifies the type of handshaking that is in effect for the SAP. ESRO_2Way specifies two-way handshaking. ESRO_3Way specifies three-way handshaking. In order for ESROS user processes to interact with one another over a network, they must specify local SAPs that use the same type of handshaking. Furthermore, once a SAP is created the handshaking type stays in effect until the SAP is released. Once an ESRO-SAP has been activated, the user process can use the services provided by ESROS sublayer.

The function returns zero if successful, otherwise it returns a nonzero error value.

3.1.3 Deactivate ESROS Service Access Point

The ESRO_sapUnbind function deactivates the ESROs service access point which is currently in use. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_sapUnbind(ESRO_SapSel sapSel)
```

The argument is defined as follows:

```
sapSel  SAP selector
```

sapSel identifies the ESROS SAP which is already in use.

The function would return 0 if successful, and a nonzero error value otherwise.

3.1.4 ESROS Invoke Service Request

The ESRO_invokeReq function requests a remote operation. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_invokeReq( ESRO_InvokeId*invokeId,/* out */
ESRO_UserInvokeRefuserInvokeRef,
ESRO_SapDesclocSapDesc,
ESRO_SapSelremESROSap,
T_SapSel*remTsap,
N_SapAddr*remNsap,
ESRO_OperationValueopValue,
ESRO_EncodingTypeencodingType,
IntparameterLen,
Byte*parameter)
```

The input arguments are defined as follows:

```
invokeId            Return value: invocation identifier
userInvokeRef       User's invocation reference
locSapDesc          The local SAP descriptor
remESROSap          Remote network SAP address
remTsap             Remote Transport SAP.
remNsap             The remote SAP selector
opValue             Operation value
encodingType        Encoding type
parameterLen        The length of the parameter
parameter           The address of the parameter buffer.
```

invokeId is assigned by ESROS sublayer. It is returned by ESROS sublayer. This identifier is used in future communications between ESROS sublayer and service user to identify the invocation for ESROS sublayer.

userInvokeRef is assigned by ESROS user. It is passed to ESROS sublayer by the user of service. This identifier is used in future communications between ESROS sublayer and service user to identify the invocation for the user of ESROS.

locSapDesc is the local SAP descriptor which is provided by ESROS sublayer at the time of SAP bind.

If ESROS can serve the invoker, the function returns 0 and the invocation identifier is returned through the invokeId parameter. If ESROS cannot serve the invoker, the function returns a nonzero failure reason value.

3.1.5 ESROS Result Service Request

The ESRO_resultReq function is issued by the performer of the operation. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_resultReq( ESRO_InvokeIdinvokeId,
ESRO_UserInvokeRefuserInvokeRef,
ESRO_EncodingTypeencodingType,
IntparameterLen,
Byte*parameter)
```

The input arguments are defined as follows:

```
invokeId        Invocation Identifier.
userInvokeRef   User's invocation reference
encodingType    Encoding type
parameterLen    Length of the parameter
parameter       Address of the parameter buffer.
```

This primitive should be issued after an ESRO_INVOKEIND event. If ESROS cannot serve the requestor, the function returns a nonzero reason value which is the failure value.

3.1.6 ESROS Error Service Request

The ESRO_errorReq function is issued by the performer of the operation in case of error in performing the operation. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_errorReq(  ESRO_InvokeIdinvokeId,
ESRO_UserInvokeRefuserInvokeRef,
ESRO_EncodingTypeencodingType,
ESRO_ErrorValueerrorValue,
IntparameterLen,
Byte*parameter)
```

The input arguments are defined as follows:

```
invokeId        The Invocation Identifier.
userInvokeRef   User's invocation reference
encodingType    Encoding type
errorValue      Identifies the nature of the error.
parameterLen    The length of the parameter
parameter       String describing the error.
```

This primitive should be issued after a INVOKEIND event. If esros cannot serve the requestor, the function returns a negative value which is the failure value.

3.1.7 Get an event

If any event has occurred in ESROS sublayer, the *ESRO_getEvent* function gets the event(s). Based on the value of *wait*, it either waits for and event (if no event available) or immediately returns.

```
PUBLIC ESRO_RetVal ESRO_getEvent( ESRO_SapDesc sapDesc,
ESRO_Event *event, Bool wait)
```

The input arguments are defined as follows:

```
sapDesc Return value: the SAP descriptor
event   ESROS event
wait    Blocking/non-blocking flag
```

The function returns any of the following event codes as the corresponding events are detected:

```
ESRO_INVOKEIND  Remote user is requesting an operation
ESRO_FAILUREIND Operation has failed
ESRO_RESULTIND  ESRO-RESULT-PDU recieved
ESRO_ERRORIND   ESRO-ERROR-PDU received
ESRO_RESULTCNF  ESROS RESULT confirm
ESRO_ERRORCNF   ESROS ERROR confirm
```

The function returns negative error number if unsuccessful, or the number of events (0 or greater than 0).

The data structures of ESROS events and the corresponding event codes are listed below:

3.1.8 Sample Code

The code fragments described in the following sections illustrate the steps required to create a ESRO service access point, and invoke and perform an operation. They are patterned after the primitives of the time sequence in , Example of time sequence diagram for ESROS Services. The code fragments themselves are listed in , ESRO API Example Usage. The code sample "invoker.c" implements the left side, and the code sample "performer.c" implements the right side.

3.1.8.1 invoker.c

invoker.c first establishes a SAP, then issues an ESRO_invokeReq of a shell command operation. In this example, the command operation is "date". It receives a confirmation (ESROESRO_ResultInd) indicating that the operation was performed. It then retrieves the results which are communicated through the ESRO_ResultInd.

3.1.8.2 performer.c

performer.c receives the ESRO_InvokeInd of a ”date” command operation in the struct ESRO_InvokeInd. The result of the command is the system date which is returned to invoker.c through ESRO_resultReq. performer.c then waits for the next request from invoker.c.

3.2 ESROS With Callback API

This section provides information about the callback API functions.

The services provided by the ESROS are defined in the ESROS Protocol Specification,”RFC-2188” [1]. The requests are issued through function calls. Callback functions associated with ESROS events are passed to ESROS at the time of sapBind function call.

The following subsections describe the ESROS library functions

3.2.1 Initialize the Parameters

```
PUBLIC ESRO_RetVal
ESRO_CB_init (String configFileName)
```

The argument is defined as follows:

```
configFileName  Config file name
```

configFileName specifies the config file name that contains ESROS initialization parameters.

3.2.2 Activate ESROS Service Access Point

The ESRO_CB_sapBind function binds an ESRO Service Access Point (ESRO_SAP) for the current user process. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_CB_sapBind(
ESRO_SapDesc    *sapDesc,
ESRO_SapSel     sapSel,
ESRO_FunctionalUnit     functionalUnit,
int (*invokeInd)(       ESRO_SapDesc    locSapDesc,
ESRO_SapSel     remESROSap,
T_SapSel        *remTsap,
N_SapAddr       *remNsap,
ESRO_InvokeId   invokeId,
ESRO_OperationValue     opValue,
ESRO_EncodingType       encodingType,
DU_View parameter),
int (*resultInd) (      ESRO_InvokeId   invokeId,
ESRO_UserInvokeRef      userInvokeRef,
ESRO_EncodingType       encodingType,
DU_View parameter),
int (*errorInd) (       ESRO_InvokeId   invokeId,
ESRO_UserInvokeRef      userInvokeRef,
ESRO_EncodingType       encodingType,
ESRO_ErrorValue errorValue,
DU_View parameter),
```

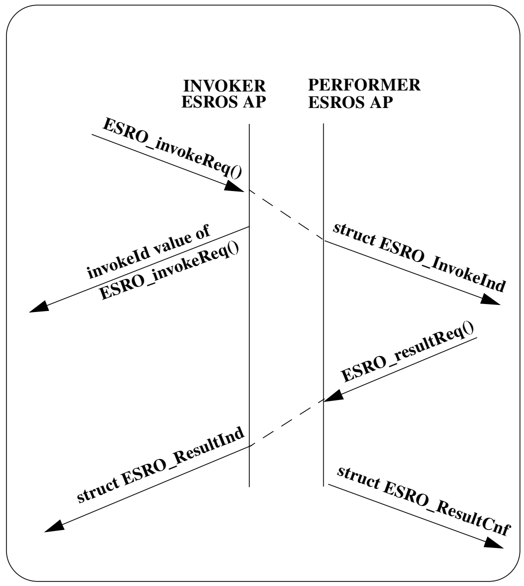

Figure 3.1: Example of time sequence diagram for ESROS Services

```
int (*resultCnf)(        ESRO_InvokeId   invokeId,
ESRO_UserInvokeRef       userInvokeRef),
int (*errorCnf) (        ESRO_InvokeId   invokeId,
ESRO_UserInvokeRef       userInvokeRef),
int (*failureInd)(       ESRO_InvokeId   invokeId,
ESRO_UserInvokeRef       userInvokeRef,
ESRO_FailureValue        failureValue))
```

The input arguments are defined as follows:

```
sapDesc Local SAP descriptor (outgoing param)
sapSel  Local SAP selector
functionalUnit  Handshaking type
locSapDesc      Local SAP descriptor
remESROSap      Remote network SAP address
remTsap Rmote Transport SAP.
remNsap The remote SAP selector
invokeId        Invocation identifier
userInvokeRef   User's invocation reference
opValue Operation value
encodingType    Encoding type
errorValue      Error value
failureValue    Failure value
parameter       parameter.

(*invokeInd)()  Invoke indication function
(*resultInd)()  Result indication function
(*errorInd)()   Error indication function
(*resultCnf)()  Result confirmation function
(*errorCnf)()   Error confirmation function
(*failureInd)() Failure indication function
```

sapDesc is a pointer to an ESRO_SapDesc structure that is created for the current user.

sapSel identifies the ESROS SAP. If the SAP is in use by another user the function returns an error value.

functionalUnit specifies the type of handshaking that is in effect for the SAP. ESRO_2Way specifies two-way handshaking. ESRO_3Way specifies three-way handshaking. In order for ESROS user processes to interact with one another over a network, they must specify local SAPs that use the same type of handshaking. Furthermore, once a SAP is created the handshaking type stays in effect until the SAP is released. Once an ESRO-SAP has been activated, the user process can use the services provided by ESROS.

After its ESRO-SAP has been activated, the user process can use the services provided by ESROS.

The function returns zero if successful, otherwise it returns a nonzero error value.

3.2.3 Deactivate ESROS Service Access Point

The ESRO_CB_sapUnbind function deactivates the ESROs service access point which is currently in use. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_sapUnbind( ESRO_SapSel     sapSel)
```

The argument is defined as follows:

```
sapSel  SAP selector
```

sapSel identifies the ESROS SAP which is already in use.
The function would return 0 if successful, and a nonzero error value otherwise.

3.2.4 ESROS Invoke Service Request

The ESRO_CB_invokeReq function requests a remote operation. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_CB_invokeReq(      ESRO_InvokeId   *invokeId,/* out */
ESRO_UserInvokeRef      userInvokeRef,
ESRO_SapDesc    locSapDesc,
ESRO_SapSel     remESROSap,
T_SapSel        *remTsap,
N_SapAddr       *remNsap,
ESRO_OperationValue     opValue,
ESRO_EncodingType       encodingType,
DU_View parameter)
```

The input arguments are defined as follows:

```
invokeId        Return value: invocation identifier
userInvokeRef   User's invocation reference
locSapDesc      The local SAP descriptor
remESROSap      Remote network SAP address
remTsap Rmote Transport SAP
remNsap The remote SAP selector
opValue Operation value
encodingType    Encoding type
parameter       user data
```

invokeId is assigned by ESROS sublayer. It is returned by ESROS sublayer and identifies an invocation for ESROS sublayer. This identifier is used in future communications between ESROS sublayer and service user to identify the invocation for ESROS sublayer.

userInvokeRef is assigned by ESROS user. It is passed to ESROS sublayer by the user of service. This identifier is used in future communications between ESROS sublayer and service user to identify the invocation for the user of ESROS.

locSapDesc is the local SAP descriptor which is provided by ESROS sublayer at the time of SAP bind.

parameter is a pointer to a DU_view data structure into which user data was previously copied. Refer to the Open C Platform document [2] for a discussion of the DU_ module.

If ESROS can serve the invoker, the function returns 0 and the invocation identifier is returned through the invokeId parameter. If ESROS cannot serve the invoker, the function returns a nonzero failure reason value.

3.2.5 ESROS Result Service Request

The ESRO_CB_resultReq function is issued by the performer of the operation. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_CB_resultReq(      ESRO_InvokeId   invokeId,
ESRO_UserInvokeRef      userInvokeRef,
ESRO_EncodingType       encodingType,
DU_View parameter)
```

The input arguments are defined as follows:

```
invokeId        invocation Identifier
userInvokeRef   User's invocation reference
encodingType    Encoding type
parameter       Parameter.
```

This primitive should be issued after invokeInd function is called. If ESROS cannot serve the requestor, the function returns a nonzero reason value which is the failure value.

3.2.6 ESROS Error Service Request

The ESRO_CB_errorReq function is issued by the performer of the operation in case of error in performing the operation. It has the following syntax:

```
PUBLIC ESRO_RetVal
ESRO_CB_errorReq(       ESRO_InvokeId   invokeId,
ESRO_UserInvokeRef      userInvokeRef,
ESRO_EncodingType       encodingType,
ESRO_ErrorValue errorValue,
DU_View parameter)
```

The input arguments are defined as follows:

```
invokeId        The Invocation Identifier
userInvokeRef   User's invocation reference
encodingType    Encoding type
errorValue      Error value
parameter       Parameter.
```

This primitive should be issued after invokeInd function is called. If ESROS cannot serve the requestor, the function returns a negative value which is the failure value.

3.2.7 Sample Code

The code fragments described in the following sections illustrate the steps required to create a ESRO service access point, and invoke and perform an operation. They are patterned after the primitives of the time sequence in , Example of time sequence diagram for ESROS CB Services. The code fragments themselves are listed in , ESRO API Example Usage. The code sample "invoksch.c" implements the left side, and the code sample "perfsch.c" implements the right side.

3.2.7.1 invoksch.c

invoksch.c first establishes a SAP, then issues an ESRO_invokeReq of a shell command operation. In this example, the command operation is "date". The resultInd function is called indicating that the operation was performed and the result is passed to it through data parameter.

3.2.7.2 perfsch.c

perfsch.c establishes a SAP and waits for a request from invoksch.c. The invokeInd function is called when the request for a command operation arrives. The result of the "date" command is the system date. perfsch.c then returns the data to invoksch.c through ESRO_resultReq. perfsch.c then waits for the next request from invoksch.c.

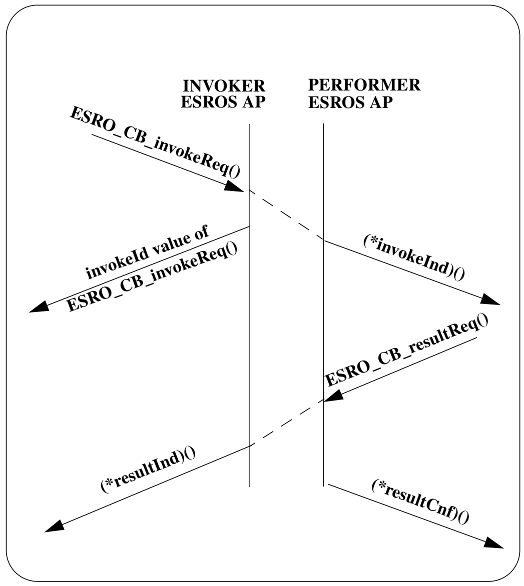

Figure 3.2: Example of time sequence diagram for ESROS CB Services

Chapter 4

ESROS Programs

4.1 Introduction

This chapter describes the design and operation of real-world ESROS programs which have been ported to both Unix and MS-DOS platforms. These programs fall into one of three categories:

- ESROS Service Provider - There is only one ESROS service provider program, the Unix ESROS-Daemon. Under MS-DOS the ESROS service provider entity is bound to individual ESROS Service Users at build-time.
- ESROS Service Users - As part of the Reference Implementation, several ESROS Service User programs have been developed to facilitate testing of ESROS installations.
- ESROS Support Programs - This category includes a program for interpreting and displaying ESROS PDU logs.

4.2 Providers and Users - Unix vs. MS-DOS/Windows

4.2.1 Unix

The ESROS provider program, ESROS, runs under Unix as a process separate from its potentially numerous user processes. There may be one or more ESROS user processes that invoke ESROS primitives via the interprocess communication mechanisms provided by the underlying operating system.

In order to run an ESROS user program under Unix, it is necessary to first start the ESROS provider program. Each user, in turn, opens an ESROS service access point (SAP) at run time through which it invokes ESROS primitives.

Each of the ESROS test utilities is really a pair of programs - one Invoker and one Performer. In order to successfully test ESROS both need to be started in the proper order: Performer first, then Invoker.

The Invoker and Performer need not share the same instance of ESROS, although it is always worthwhile to test ESROS in this mode. Indeed, it is perfectly normal to have the Invoker and its copy of ESROS execute on one machine and the Performer and its copy on another (provided, of course, that the two machines have IP connectivity). These two cases are illustrated in the above figure. In either case, the Invoker application must specify the IP address of the peer Performer/ESROS combination. This is discussed in greater detail in later sections of this chapter.

4.2.2 MS-DOS/Windows

Under MS-DOS, ESROS users are linked to the ESROS provider at build-time, since MS-DOS does not provide facilities for running multiple, concurrent processes. In the case of Microsoft Windows, the same single process model is used. Thus, at any given time, only one application can be running. In the case of the test programs this will be either an Invoker or a Performer. That application's peer must execute on another network-connected machine (either Unix or MS-DOS or Windows) in order to execute successfully.

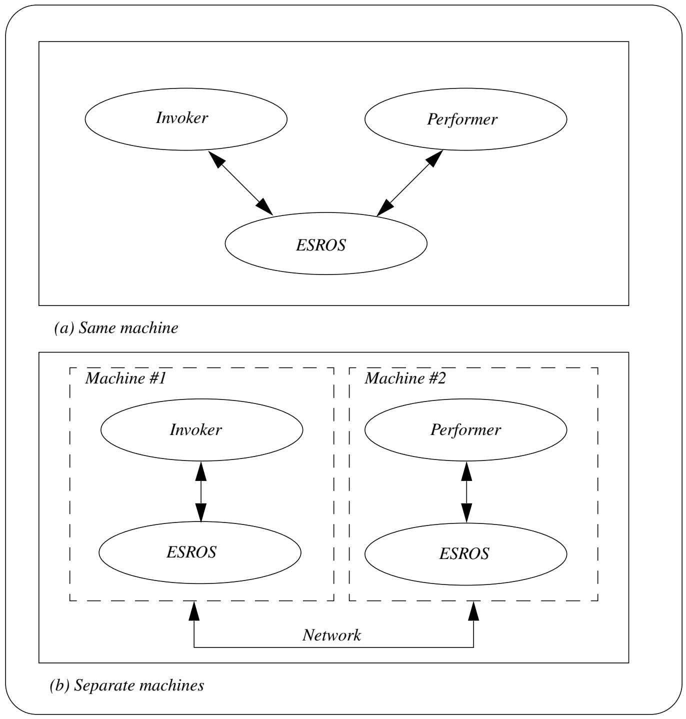

Figure 4.1: Unix ESROS Processes.

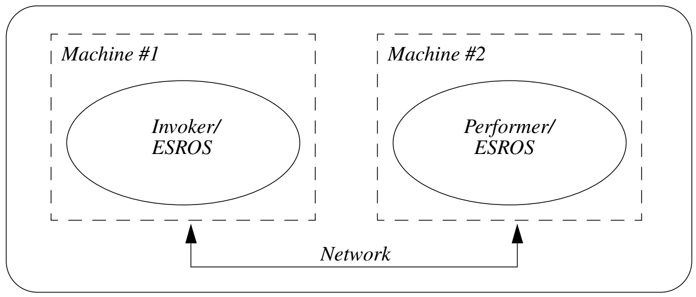

Figure 4.2: MS-DOS/Windows ESROS Processes

4.3 The ESROS Service Provider

There is just one ESROS service provider program - the Unix ESROS-Daemon. Under MS-DOS and Windows the ESROS service provider entity is linked with the various ESROS users at build-time.

4.3.1 Unix

```
esros [ -p port ] [ -s socket ] [ -c configfile ]
[ - o outlogfile ] [ -T module,mask ... ] ...
```

This command is described in detail in its man page. See Appendix C.

The following listing describes the format of the esros configuration file which is specified with the -c command line option. In the absence of the -c option esros looks for the file esrop.ini in the same directory as the esros binary. Note that command line options, when present, override their configuration file counterparts.

4.3.1.1 Running esros under Unix

First go to the root directory of the ESROS source tree and execute the following command:

```
source sourceme.csh
```

Then go to the directory containing shell scripts for running programs and start ESROS using the shell script runEsros.sh. You may first wish to edit this script to enable tracing or PDU logging. (See Section 4.8 for more information on tracing and logging.) Do this for both machines if you are running the Invoker and Performer in isolation.

The shell script will prompt you to take further action to display the trace output. This involves running the Unix tail utility on the trace file. You may also wish to display either of the PDU transaction logs by running esropscop as described in Chapter 4.

Next, proceed to one of the following three sections for instructions on how to run the test itself.

4.3.2 MS-DOS/Windows

As discussed previously, the ESROS provider does not execute as a separate task under MS-DOS/Windows. Rather, it is linked at build-time with the various user programs. However, the command line options for the Unix version still apply. They are simply entered on the command line of the user program along with that program's specific options. Again, refer to the esros man page in Appendix C for a discussion of the command line options for esros. The MS-DOS/Windows ESROS provider also uses the esros configuration file in the same manner as the Unix implementation.

4.4 Service User Programs

This section describes the ESROS user programs that were developed to support testing of ESROS installations.

There are three groups of test utilities that are ported to both Unix and MS-DOS/Windows. Under Unix, these are known as ops_xmpl (single operation invoker/performer), stress (Stress Tester), and tester (ESROS Scenario Interpreter).

- ops_xmpl - quick test, a simple/single operation
- stress - stress tester, repetitive operations (stress_i/stress_p)
- tester- scenario interpreter, complex transactions (esrossi)

Under MS-DOS/Windows these utilities are linked to the ESROS provider program at build-time. (These so-called stand alone versions of the user programs also exist under Unix for test purposes.)

4.4.1 ops_xmpl

ops_xmpl invoker and performer programs invoke and perform a single operation respectively.

4.4.1.1 Unix

Single Operation Invoker:

```
invoker -l localEsroSapSel -r remoteEsroSapSel -p remotePortNu
-s pubQueueName -n remoteIPadr [-T <module_name>,<bit_mask> ] ...
```

Single Operation Performer:

```
performer -l localEsroSapSel -s pubQueueName [-T <module_name>,<bit_mask>] ...
```

4.4.1.2 MS-DOS/Windows

Single Operation Invoker:

```
invoker -l localEsroSapSel -r remoteEsroSapSel -p remotePortNu
-s pubQueueName -n remoteIPadr -o outfile -e errfile
[-T <module_name>,<bit_mask>] ...
```

Single Operation Performer:

```
performer -l localEsroSapSel -s pubQueueName -o outfile -e errfile
[-T <module_name>,<bit_mask>] ...
```

4.4.2 stress

stress is very similar to ops_xmpl except that it repetitively invokes and performs the same operation every delay milliseconds. Also, the invoker performs cumulative error reporting every reportPeriod milliseconds to outfile, which defaults to stdout and can be set to a file. See the previous section for a discussion of the other options.

4.4.2.1 Unix

Stress Tester Invoker:

```
stress_i -l localEsroSapSel -r remoteEsroSapSel -p remotePortNu
-s -o logfile pubQueueName -n remoteIPadr -d delay -t reportPeriod
-o logFile [-T <module_name>,<bit_mask>] ...
```

Stress Tester Performer:

```
stress_p -l localEsroSapSel -s pubQueueName [-T
<module_name>,<bit_mask>] ...
```

4.4.2.2 MS-DOS/Windows

Stress Tester Invoker:

```
stress_i -l localEsroSapSel -r remoteEsroSapSel -p remotePortNu -s
pubQueueName -n remoteIPadr -d delay -t reportPeriod -o logFile
[-T <module_name>,<bit_mask>] ...
```

Stress Tester Performer:

```
stress_p -l localEsroSapSel -s pubQueueName [-T <module_name>,<bit_mask>] ...
```

4.4.3 Tester

Tester (also called esrossi) is the ESROS scenario interpreter. The same program is used as both the Invoker and the Performer. The program reads scenario files that contain sequences of commands that result in ESROS primitive invocations. The scenarios themselves are discussed in greater detail in the next sections.

4.4.3.1 Unix

```
esrossi [-d scenariosDir] [-T <module_name>,<bit_mask>] ...
```

4.4.3.2 MS-DOS/Windows

```
esrossi [-d scenariosDir] [-o outfile] [-e errfile]
[-T <module_name>,<bit_mask>] ...
```

4.5 Support Programs

4.5.1 esropscop

This Program is used to interpret and display the ESROS protocol log file.

```
esropscop esroslog.pdu
```

4.6 Running the Test Programs

4.6.1 Unix

4.6.1.1 ops_xmpl

Go to the directory containing the shell scripts for running programs. There you will find the shell scripts runExamplePer.sh and runExampleInv.sh . These scripts are used to execute single operation performer and invoker, respectively. Edit these files to enable any desired tracing options. (Do this on both machines if running the Invoker and Performer in isolation.) Also, be sure to specify the correct remoteEsroSapSel and remoteIPAdr in runExampleInv.sh. For example,

```
#
#runExampleInv.sh
#
$BINDIR/bin/invoker -l 12 -r 15 -p 2002 -s /tmp/SP
-n 198.62.92.28 -T G_,ffff  -T ESRO_,ffff  -T IMQ_,ffff
-T SCH_,ffff -T FSM_,ffff

#
#runExamplePer.sh
#
$BINDIR/bin/performer -l 15 -s /tmp/SP -T G_,ffff
-T ESRO_,ffff  -T IMQ_,ffff
```

In this case 198.62.92.28 is the IP address of the machine on which the Performer and its copy of ESROS will execute.

Also, note that the remote and local SAP selectors of the Invoker and Performer, respectively, match, i.e. the -l 15 option in the Performer corresponds to the -r 15 option in the Invoker.

Once the shell scripts are correct, execute the Performer script and then the Invoker.

4.6.1.2 stress

Go to the directory containing the shell scripts for running programs. There you will find the shell scripts runStressPer.sh and runStressInv.sh. These scripts are used to execute stress_p and stress_i, respectively. Edit these files as necessary. Start the Performer and then the Invoker.

4.6.1.3 esrossi

As stated earlier, there is just one esrossi program which serves as both invoker and performer, depending on the contents of the scenario file given to it. The scenario files are located in the scenarios directory. For each scenario there are two files, an invoker and a performer. For instance, inv1001 and perf1001 are invoker and performer scenarios respectively. For each scenario that you wish to run, you must edit the Invoker file, changing the remote IP address to match that of the machine on which you will execute the Performer script. In most of the scripts you must change the IP address in multiple locations.

Once you have edited the scenario files, go to the directory containing the shell scripts for running programs. You may edit the shell script runEsrossi.sh to specify any trace options that you wish. Then, start the Performer and the Invoker using the shell script. In both cases you will be prompted to enter the name of the scenario file. Enter the full path name of the file, Performer first (perfxxxx) and then Invoker (invxxxx).

4.6.2 MS-DOS/Windows

The MS-DOS/Windows versions of the test utilities behave in a manner very similar to that of their Unix counterparts. However, as noted above, the test utilities are bound to ESROS at build time and the Invoker and Performer cannot both execute simultaneously on the same machine. So, it is not necessary to first start ESROS. Simply run the application program. It is still necessary, however, to start the Performer first and then the Invoker.

The MS-DOS/Windows test utilities will interact with their Unix counterparts. So you may run a given Performer on a Unix machine and the Invoker on an MS-DOS/Windows machine, or visa-versa.

4.6.2.1 esros

Since ESROS is bound to the utilites, ESROS tracing and PDU logging are controlled from the utility's command line (and ini file).

4.6.2.2 ops_xmpl

Go to the group file containing icons for esros programs. There you will find icons that can run invoker and performer programs. Edit these as you would under Unix. Note, however, that you may also specify the ESROS transaction log file names here.

Start the Performer on one machine and the Invoker on another.

4.6.2.3 stress

Go to the group file containing icons for esros programs. There you will find icons that can run invoker and performer programs. Edit these as you would under Unix. Note, however, that you may also specify the ESROS transaction log file names here.

Start the Performer on one machine and the Invoker on another.

4.6.2.4 tester

The scenario files are located in the scnarios directory. Go to this directory and edit as you would under Unix.

Once you have edited the scenario files click on the scenario interpreter icon. You may edit the icon or ini file to specify any trace options that you wish. Then, start the Performer on one machine and the Invoker on another. In both cases you will be prompted to enter the name of the scenario file. Enter the full path name of the file, Performer first (perfxxxx) and then Invoker (invxxxx).

4.7 Senarios

This section describes the format of an ESROS Scenario Interpreter (esrossi) scenario file.

Scenario files are normally written in pairs, one for the invoker and one for the performer. Scenario files are plain files containing one or more commands which are described in detail in the following sections. Some of the commands are used exclusively by performers or by invokers. Other commands may be used by either.

Scenarios are intended to be serial in nature with the major commands grouped in matching pairs. This means that when an invoker issues an invoke request command, for instance, the performer should have an outstanding invoke indication command whose parameters match those of the invoker's. A mismatch either in the parameters or in the type of the event expected by the performer results in a performer error.

When reading this section refer to ESRO Testing for more information.

4.7.1 Logging-Related Commands

Logging-related commands generate log messages and control the display of messages to the screen and the writing of messages to files. These commands are used by both performers and invokers.

4.7.1.1 log

```
log <message>

     message   quoted string

DESCRIPTION

log prints log messages to stdout and/or an optional log file.

EXAMPLE

log "making an invoke request now"
```

4.7.1.2 logfile

```
logfile <filename>

     filename   quoted string

DESCRIPTION

logfile causes log messages to be copied to filename

EXAMPLE
```

```
logfile "1001.log"
```

4.7.1.3 quiet

```
quiet

DESCRIPTION

quiet disables the display of log messages on the screen.
However, if a log file is open, messages are still written to it.
```

4.7.1.4 verbose

```
verbose

DESCRIPTION

verbose enables the display of log messages on the screen.
If a log file is open messages are still written to it.
```

4.7.2 Invocation-Related Commands

4.7.2.1 invoke request

```
invoke request <remsap> <remport> <ipaddr> <operationval> <encodetype> <data>

   remsap          number
   remport         number
   ipaddr          dotted quad notation
   operationval    number
   encodetype      number
   data            quoted string

USED BY: invoker

DESCRIPTION

invoke request is used by an invoker to generate an
ESROS-INVOKE.request. The ESROS-INVOKE PDU is sent to a performer
listening to remsap. The performer verifies that operationval,
encodetype, and data match the values which it expects using the
invoke indication command, below. The esros provider associated with
the performer is located at ipaddr remport. Note that this is the only
command in which the IP address or remote port of either performer or
invoker are referred to. In all subsequent transactions relating to
this invoke request, the address and port are retained and reused by
the scenario interpreter.

EXAMPLE
```

```
invoke request 13 2002 198.62.92.10 2 0 "date"
```

4.7.2.2 invoke indication

```
invoke indication <operationval> <encodetype> <data>

     operationval     number
     encodetype       number
     data             quoted string
```

USED BY: performer

DESCRIPTION

invoke indication is used by a performer to receive an ESROS-INVOKE.indication. The operation value, encoding type, and data of the received ESROS_INVOKE PDU must match the operationval, encodetype, and data, respectively, of the command in order for the command to execute successfully. When a mismatch error occurs the performer logs an error message. Also, if a different event is received the program logs an error message.

EXAMPLE

```
invoke indication 2 0 "date"
```

4.7.3 Result-Related Commands

4.7.3.1 result request

```
result request <encodetype> <data>

     encodetype     number
     data           quoted string
```

USED BY: performer

DESCRIPTION

result request is used by a performer to issue an ESROS-RESULT.request in response to the most recently received ESROS-INVOKE.indication. The scenario interpreter uses the most recently received invocation ID to generate the ESROS-RESULT PDU. The invoker verifies that encodetype and data match the values which it expects using the result indication command, below.

EXAMPLE

```
result request 0 "October 22, 1995"
```

4.7.3.2 result indication

```
result indication <encodetype> <data>

     encodetype   number
     data         quoted string
```

DESCRIPTION

result indication is used by an invoker to receive an ESROS-RESULT.indication event. The encoding type, and data of the received ESROS-RESULT PDU must match the encodetype and data, respectively, of the command in order for the command to execute successfully. When a mismatch error occurs the performer logs an error message. Also, if a different event is received the program logs an error message.

EXAMPLE

```
result indication 0 "October 22, 1995"
```

4.7.3.3 result confirmation

```
result confirmation
```

USED BY: performer

DESCRIPTION

result confirmation is used by a performer to receive an ESROS-RESULT.confirm event. If a different event is received the performer logs an error message.

EXAMPLE

```
result confirmation
```

4.7.4 Error-Related Commands

4.7.4.1 error request

```
error request <encodetype> <errorvalue> <data>

     encodetype   number
     errorvalue   number
     data         quoted string
```

USED BY: performer

DESCRIPTION

error request is used by a performer to send an ESROS-ERROR.request in response to the most recently received ESROS-INVOKE.indication. The scenario interpreter uses the most recently received invocation ID to generate the ESROS-ERRO PDU. The invoker verifies that encodetype, errorvalue, and data match the values which it expects using the error indication command, below.

EXAMPLE

```
error request 0 2 "Unknown encoding type"
```

4.7.4.2 error indication

```
error indication <encodetype> <errorvalue> <data>

     encodetype   number
     errorvalue   number
     data         quoted string
```

USED BY: invoker

DESCRIPTION

error indication is used by an invoker to receive an ESROS-ERROR.indication event. The encoding type, error value, and data of the received ESROS-ERROR PDU must match the encodetype, errorvalue, and data, respectively, of the command in order for the command to execute successfully. When a mismatch error occurs the invoker logs an error message. Also, if a different event is received the program logs an error message.

EXAMPLE

```
error indication 0 "Unknown encoding type"
```

4.7.4.3 error confirmation

```
error confirmation
```

USED BY: performer

DESCRIPTION

error confirmation is used by a performer to receive an error ESROS-ERROR.confirm event. If a different event is received the performer logs an error message.

EXAMPLE

```
error confirmation
```

4.7.5 General Commands

These commands may be used either by an invoker or performer.

4.7.5.1 sapbind

```
sapbind <localsap> <func_unit>

     localsap     number
     func_unit    2 | 3

DESCRIPTION

sapbind is used by a performer or an invoker to establish a local
service access point (SAP) with ESROS.  localsap specifies the SAP
number. func_unit specifies the type of handshaking that is in effect
for the SAP. 2 specifies two-way handshaking. 3 specifies three-way
handshaking. Note that in order for an invoker and performer to
interact they must do so via local SAPs that use the same type of
handshaking. Furthermore, once a SAP is created the handshaking type
stays in effect until the SAP is released.

EXAMPLE

sapbind 12
```

4.7.5.2 saprelease

```
saprelease <localsap>

     localsap     number

DESCRIPTION

saprelease is used by a performer or an invoker to release an ESROS
service access point that was previously opened by a sapbind command.

EXAMPLE

saprelease 12
```

4.7.5.3 failure indication

```
failure indication <value>

     value     number

DESCRIPTION

failure indication may be used by either a performer or an invoker to
receive an ESROS-FAILURE.indication event. The reported error value
```

must match value in order for the command to execute
successfully. When a mismatch occurs the program logs an error
message. If a different event is received the program logs an error
message. value may be any one of the following tokens:

```
       TRANSFAIL
       LOCRESOURCE
       USERNOTRESP
       REMRESOURCE
```

EXAMPLE

```
failure indication TRANSFAIL
```

4.7.5.4 rawevent

```
rawevent <eventnumber>

     eventnumber     number
```

DESCRIPTION

rawevent checks to see if an arriving event is of type eventnumber. If
a mismatch occurs the program generates an error message to the
log. eventnumber may be one of the following:

```
/* Event Codes */
#define  ESRO_E_BASE     200
#define ESRO_INVOKEIND  (ESRO_E_BASE+0)
#define ESRO_RESULTIND  (ESRO_E_BASE+1)
#define  ESRO_ERRORIND   (ESRO_E_BASE+2)
#define  ESRO_RESULTCNF  (ESRO_E_BASE+3)
#define ESRO_ERRORCNF   (ESRO_E_BASE+4)
#define ESRO_FAILUREIND (ESRO_E_BASE+5)
```

EXAMPLE

```
rawevent 202
```

4.7.5.5 delay

```
delay <seconds>

     seconds     number
```

DESCRIPTION

delay is used by a performer or an invoker to pause for seconds before
executing the next command in the script.

EXAMPLE

```
delay 2
```

4.7.6 Scenario File Manipulation Commands

4.7.6.1 include

```
include <filename>

     filename     quoted string

DESCRIPTION

include is used by a performer or invoker to include text from another
file, much like a C language #include directive.

EXAMPLE

include "perf2003"
```

4.7.6.2 path

```
path <name>

path     quoted string

DESCRIPTION

path sets the name of the default directory for scenario files. This
directory is used for any subsequent include commands. The path
command overrides the equivalent esrossi command line argument.

EXAMPLE

path "/lib/scenarios"
```

4.8 Tracing

ESROS programs use two different types of tracing in order to facilitate debugging and integration. The first type of tracing is the Open C Platform (OCP) Trace Module tracing. The second type is physical data unit (PDU) transaction logging used exclusively by the provider program. (Remember, however, that under MS-DOS, the provider and user programs are one and the same. So a given test program under MS-DOS will have both types available.)

4.8.1 OCP Trace Module Tracing

The Trace Module (TM) selectively and dynamically generates trace messages in order to facilitate program debugging. Within OCP, each module may have its own particular degree of TM tracing enabled at run-time. Some ESROS-specific modules also use TM tracing in a similar manner.

Tracing may also be globally enabled or disabled at compile time without performing any changes to the source code, other than changing the value of a defined value in a single include file. This allows fully debugged programs to occupy minimal space.

For a more complete discussion of the Trace Module and of OCP see the Open C Platform document [2].

4.8.1.1 Run Time Control of TM_

When starting an ESROS program, if you wish to enable trace options for a module, you should specify on the command line the following:

```
-T <module_name>,<hex_bits>
```

where ¡module_name¿ is a valid OCP or ESROS module and ¡hex_bits¿ is a hexadecimal value specifying which trace bits for that module to enable. Multiple sets of -T options may be specified to enable tracing for more than one module. For example:

```
esros -T LOPS_,ffff -T IMQ_,3
```

General usage dictates that lower-order bits produce less output. The lowest-order bits are often useful even during normal operation of the application. Bits above the first byte are reserved for trace options that provide a lot of output, such as dumping of complete PDUs.

4.8.1.2 TM Output

The following example illustrates the format of a trace message. Each trace message contains the source file name and line number from which the message was generated, followed by a variable number of user data fields.

```
clinvktd.c, 197: fsm_ePass: machine=0x60e38  evtId=0x8
```

4.8.2 PDU Transaction Logging

In PDU transaction logging, each ESRO network transaction performed by the ESROS provider is logged to one of two files in a compressed, binary format that may be subsequently displayed using a special purpose display utility. The first of the two files contains the history of all transactions while the second contains only those that indicate the occurrence of some sort of error.

PDU transaction logging is enabled on the ESROS command line in the following manner:

```
esros [-o <all_file_name>] [-e <error_file_name>]
```

Note that either type of logging is purely optional. Thus, the -o option could be used during a debug session while the -e option might be used to log spurious errors that occur during normal operations.

4.8.2.1 PDU Transaction Log Display

PDU transaction logs may be displayed using the utility esropscop in the following manner:

```
esropscop -f <file_name>
```

A sample output from this program is shown below.

```
SYSENV=/h9/neda/sw/curenv.sol2/results/systems/arash

----------------------------------------------------------------------
  Time   TSDU Tsiz  Loc    Rem Ref  Dst  Src  OpVal Encod  Parameter
----------------------------------------------------------------------
00:30.2    1    7      <- inv   1   12   13    2     2    ...date
```

The contents of this file are described in appendix D ESRO Program man Pages (esropscop man page).

4.9 Implementation Notes

This section addresses implementation-specific aspects of the ESROS programs.

4.9.1 Differences Between Unix and MS-DOS Portations

The major differences between the Unix and MS-DOS/Windows portations are isolated to several discrete locations in the code. These differences are discussed in the following sections.

4.9.1.1 Multiple Processes vs. Single Process

The most significant difference between the two portations lies in the manner in which an ESROS user communicates with ESROS itself. Under Unix, ESROS runs as a separate task. There may thus be one or more ESROS users that invoke ESROS primitives via an interprocess communication mechanism provided by the UPQ_BSD_ module.

MS-DOS, of course, does not support such a mechanism. Therefor, ESROS and ESROS users are linked together as single executable. This is the same for Windows. As explained in the previous sections, ESROP has a call-back interface to the upper layer. However, there are two different API's available to the ESROS user (see Section, ESROS API). In the case of applications that use the Call-back API, the inter-process communication module is eliminated for one-process model. In the case of Function Call API, in place of the interprocess communication mechanism, there is a a module that emulates the UPQ_BSD_ facilities. This module, named UPQ_SIMU_, uses disk files to simulate the interprocess communication mechanism.

The developer of an ESROS application who has to port between these two environments will be chiefly concerned with the ESROS user's make file. Under Unix and for two process model, the ESROS user application is linked to the UPQ_BSD_ library. Under MS-DOS it is linked to the UPQ_SIMU_ library. In addition, the MS-DOS user must link to the libraries containing the ESROS code. The following excerpts illustrate this principle.

Libraries used building Unix ESROS user application as a separate process

```
USER_SH = $(LIBS_PATH)/esro_ushcb.a
UPQ = $(LIBS_PATH)/upq_bsd.a
GF = $(LIBS_PATH)/gf.a
```

Libraries used building MS-DOS ESROS user application in one process with Function Call API

```
USER_SH = $(LIBS_PATH)\esro_ush.lib
PRVDR_SH = $(LIBS_PATH)\sp_shell.lib
UPQ_SIMU = $(LIBS_PATH)\upq_simu.lib
UDP_IF = $(LIBS_PATH)\udp_pco.lib
ESROP_SH = $(LIBS_PATH)\esrop_sh.lib
PROT_ENG = $(LIBS_PATH)\esroprot.lib
GF = $(LIBS_PATH)\gf.lib
```

Libraries used building MS-DOS ESROS user application in one process with Call-back API

```
UDP_IF = $(LIBS_PATH)\udp_pco.lib
ESROP_SH = $(LIBS_PATH)\esrop_sh.lib
PROT_ENG = $(LIBS_PATH)\esroprot.lib
GF = $(LIBS_PATH)\gf.lib
SF = $(LIBS_PATH)\sf.lib
FSM = $(LIBS_PATH)\fsm.lib
```

4.9.1.2 Scheduler Module (SCH_)

SCH_ module can be used for scheduling the program's modules.

One of the common usages of SCH_ module is scheduling of further processing within the same module. This happens most often to prevent re-entry to non-re-entrant code. For more information about the Scheduler module refer to OPEN C Platform document.

4.9.1.3 Timer Module (TMR_)

The TMR_ module defines a model and an interface for providing timer facilities to Open C Layers, regardless of the environment, provided that all implementations of the TMR_ module conform to the interface defined here. For more information about the Timer module refer to OPEN C Platform document.

4.9.1.4 FLEX & BISON

FLEX is a DOS portation of the lex utility commonly found on Unix systems. BISON is a DOS portation of the yacc utility. These utilities are used in the compilation of the ESROS Scenario interpreter, ESROSSI.

Chapter 5

ESROS Testing

This chapter describes the testing methodologies and tools used to test ESROS implementations.

5.1 Conformance and Interconnection Testing Overview

This section describes conformance testing and interconnection testing. Much of the following has been extracted from [ISO/IEC 9646].

Conformance testing has two components: static conformance tests and dynamic conformance tests. Static conformance is a paper evaluation of a layer implementation's ability to meet the conformance requirements. It uses the information provided by the implementation supplier to determine the capabilities of the implementation. For example, a manufacturer may choose to not implement an optional feature of a protocol. In such a case the manufacturer's release notes would clearly indicate this feature is not supported, and therefore tests exercising this function would not be run against the device. Dynamic conformance is the actual execution of a test suite to exercise the implementation, observing its behavior under a variety of conditions.

Interconnection tests, a subset of the Dynamic Conformance tests, are normally applied to implementations operating across a user-network boundary. They can also be used for network-to-network testing. This approach is not the same as rigorous conformance testing, as defined by the [ISO/IEC-9646] standards and as generally used in the industry. But interconnection testing does increase the likelihood of implementations from two different manufacturers successfully interoperating, both with the network and directly with each other. The difference between interconnection testing and conformance testing lies in the coverage offered by the tests run against the implementation.

Interconnection tests exercise an implementation in such a way as to cause it to exhibit some expected behavior when using the network to communicate with a peer entity. Interconnection Tests do not test:

- That the device offers the user adequate service or performance
- That the device behaves correctly during error conditions which do not affect the network, but which may impact user expectations.

5.2 Abstract Test Methods

This section summarizes the characteristics of various Abstract Test Methods (ATMs) currently recognized by [ISO/IEC-9646] and presents concerns relevant to implementation manufacturers, vendors, test suite developers and test laboratories. This section does not explore novel testing approaches nor does it consider multi-layer ATM variants.

5.2.1 What Is an ATM?

An Abstract Test Method (ATM) describes a testing architecture, consisting of a lower tester, upper tester, and test coordination procedures, and their relationships to the test system, the tester (LT), and the System Under Test (SUT). The testing architecture is a generalized model based on the Open Systems Interconnection (OSI) Reference Model layered approach to protocol architecture. Since the OSI model does not impose a particular scheme on the implementation, the test method cannot assume the availability of functions or control beyond those required by a protocol specification. Hence the testing method is an ”abstract” test method based on the OSI Reference Model. The test suite developers must ensure that a model used for developing a test suite and test cases is realizable in a real test execution environment.

An ATM is described in terms of the outputs observed from the Implementation Under Test (IUT) and inputs that can be controlled. Each ATM determines the Points of Control and Observation (PCOs) and test events (Abstract Service Primitives (ASPs) and Protocol Data Units (PDUs) to be used in an abstract test case.

5.2.1.1 The ATM/ATS Relationship

Abstract Test Suites are Abstract Test Method specific. For each ATM, there exists a unique ATS defining the PCOs, ASPs and PDUs used for observing and controlling an implementation’s behavior during testing. As an example, the Abstract Test Suite written based on the Remote Abstract Test Method differs from an Abstract Test Suite written based on the Local Abstract Test Method. One difference is that the Remote method uses one PCO at the Lower Service Boundary, whereas the Local method uses two PCOs; one at the lower service interface, and the other at the upper service interface

5.2.1.2 Applicability

The Abstract Test Methods discussed here are applicable to protocols adhering to the principles of layering as defined in the ISO OSI Reference Model [ISO-7498] and [CCITT-X.200]. The test methods identified will be considered for Interconnection and Conformance Testing of the ESROS protocols. The information regarding ATMs has been extracted from [ISO/IEC-9646].

5.2.2 Remote Test Method

The Remote test method is illustrated in Figure 5.1 and is characterized by the following:

- A single PCO located at the LT in the Test System
- Access to an Upper Tester is not formally required

Test Coordination Procedures between the Test System LT and the UT (if one exists) are implied or defined in an ad hoc manner. Typically coordination is achieved by a human user intervening at a system console.

Note that it may not always be possible to realize the required test coordination between the Test System and the SUT at the UT service boundary, however:

- It may be impossible to initiate some test cases from the LT because of Service Provider restrictions or limitations
- It is the simplest ATM
- It makes no special demands on the SUT
- It makes no assumptions about the internal design of the SUT or the IUT within Service Provider PDUs
- It essentially treats the SUT as a black box
- It is the least intrusive of the methods presented.

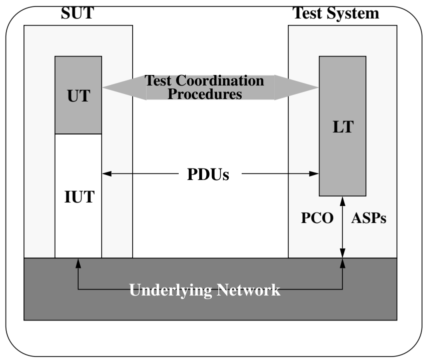

Figure 5.1: Remote Test Method

5.2.3 Coordinated Test Method

The Coordinated test method is illustrated in Figure 5.2 and is characterized by the following:

- A single PCO (located at the Lower Tester (LT) in the Test System)
- No access to an Upper Tester (UT) is required
- Test Coordination Procedures between the Test System Lower Tester and the SUT are formalized by the existence of a Test Management protocol. The protocol allows the tester to control SUT actions, events, messages and report on events coming from the UT (if one exists). To reduce complexity some have proposed sending TM-PDUs as user data of the protocol under test.

Note that even with the TM protocol, it may be difficult or impossible to realize the required test coordination between the Test System and the SUT at the UT service boundary. Service Provider PDUs

It may be impossible to initiate some test cases from the LT because of Service Provider restrictions or limitations

Additional coordination via the TM protocol implies the addition of new functionality in the SUT. A protocol independent TM protocol (specified by [ISO/IEC-9646]) does not yet exist. Use of this ATM would require the design and development of customized TM protocols for each protocol using the coordinated test method

The internal SUT design must consider the TM protocol

This method accesses the SUTs internals and could be described as glass box testing

Increases the complexity of the ATS (upper test must be tested to verify that it conforms to the TM protocol specification).

5.2.4 Distributed Test Method

The distributed test method is illustrated in Figure 5.3 and is characterized by the following:

- It has two PCOs (one at Lower Tester in the Test System the other at the Upper Tester in the SUT) Service Provider PDUs
- Access to UT (Upper service boundary of IUT) is required
- Test Coordination Procedures between the Test System LT and the UT are more rigorous. Explicit control and observation of ASPs is possible. ATS makes explicit use of these ASPs. Test Coordination Procedures require either the use of a human operator or the use of a standardized programming language interface. In most cases a human user acts as the interface between the Test System and the UT. The interface is done using a system console and/or by telephone with the remote Test System operator.

Note that it may not always be possible to realize the required test coordination between the Test System and the SUT at the UT service boundary, however:

- It may be impossible to initiate some test cases from the LT because of Service Provider restrictions or limitations
- Increases the complexity of the ATS (ASPs in the SUT must be specified in the ATS)
- Access to Upper service interface is better than that of the remote test method
- Easier to implement than the local test method (see below)
- Synchronization is difficult.

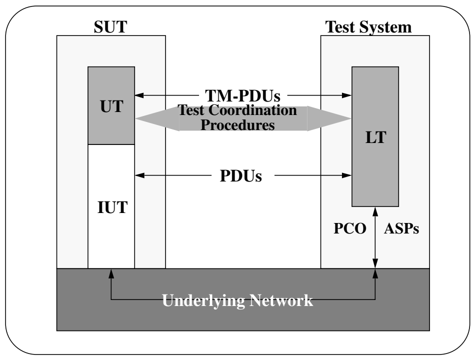

Figure 5.2: Coordinated Test Method

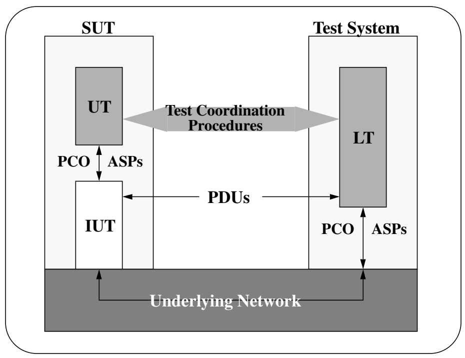

Figure 5.3: Distributed Test Method

5.2.5 Local Test Method

The Local test method is illustrated in Figure 5.4 and is characterized by the following:

- It has two PCOs (one at LT in the Test System, the other at the UT in the SUT)
- Because both the UT and LT are located in the Test System, two hardware interfaces to the SUT are required
- Additional functionality must be added to support the UT hardware interface
- Increases the complexity of the ATS
- Capable of exercising any protocol state transition (transient states excluded).

It may be impossible to initiate some test cases from the LT because of Service Provider restrictions or limitations. The Local test method is often described as having no practical application.

5.3 The ESROS Test Tools

The Neda ESROS Test Tools are a series of programs and related data files that include a scenario interpreter, a group of scenario files, an exception generator, and an exception generator scope program. Installation and operation of these programs is described in detail in ESRO Programs and in ESRO Program man Pages.

The Test Tools are designed to run on Solaris 5.4. (The scenario interpreter has also been ported to MS-DOS.) Each of the tools is implemented as a separate process. Interprocess facilities available under Solaris are used to establish channels between processes at run-time. The individual components are:

- esrossi - The scenario interpreter. An application that reads and executes scenario files.
- Scenario files - Contain descriptions of the invocations and requests that are to be sent and the indications and confirmations that the implementation expects to receive.
- lrexgen - The exception generator. A program that observes and records Protocol Data Unit (PDU) traffic between the ESROS and the Transport Program. The exception generator is also used to introduce intentional errors into PDUs that consequently produce observable results.
- esropscop - The exception generator scope program. A formatting program that processes the binary files generated by the exception generator and produces a human-readable list of the content of ESROS Protocol Data Unit traffic.

5.3.1 How the Test Tools Work

Each element of the test tools operates as a separate process, as shown in Figure 5.5

esrossi, which is run both locally and remotely in parallel, reads commands in the Test Center (local) and Implementation Under Test (remote) scenario files.

Commands in the Test Center and Implementation Under Test scenario files are synchronized so that, together, they test a feature or a set of features of the ESROS protocol. Commands in scenario files are implemented sequentially.

lrexgen, which is run locally, operates between the ESROS and Transport layers. lrexgen establishes an interprocess communication channel between itself and esrossi, which can also control the behavior of lrexgen.

5.4 Test Objectives

5.4.1 Valid sequences of primitives

Such sequence of primitives is valid based on the protocol specification. The behavior of system for valid sequence of primitives is tested.

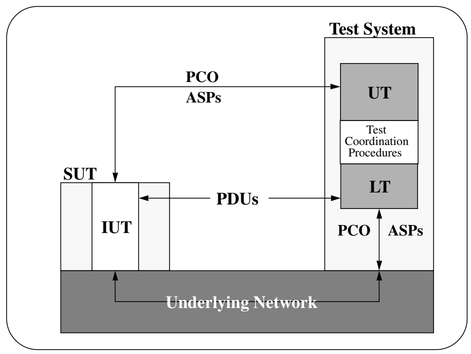

Figure 5.4: Local Test Method

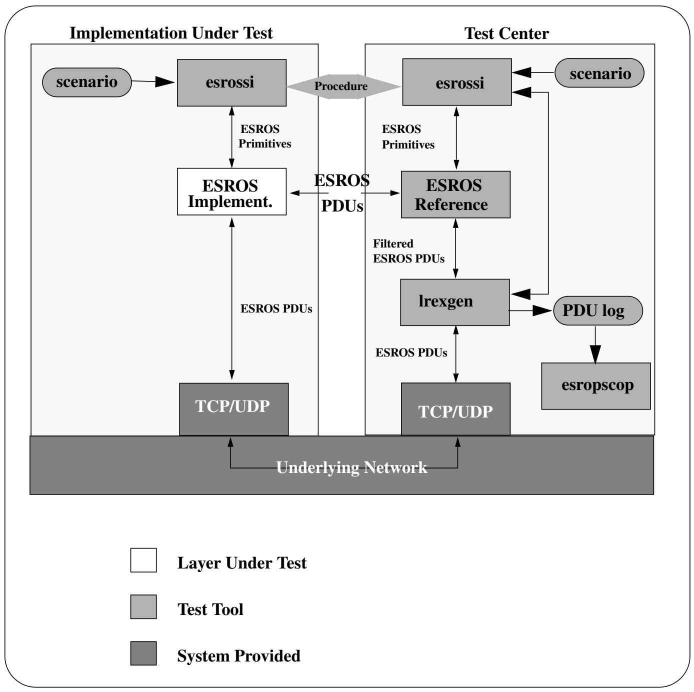

Figure 5.5: ESROS Test Tools

5.4.2 Invalid sequences of primitives

Such sequence of primitives is invalid based on the protocol specification. The behavior of system for invalid sequence of primitives is tested.

5.4.3 Parameter variations on primitives

Functionality of the system for different variations of parameters is tested.

5.4.4 Stress Tests

System is put under stress and functionality of system under stress is tested.

5.4.5 Multiple results

Multiple results is sent by performer to test the behavior of the invoker in the case of duplicate PDU's.

5.5 Test Cases

5.5.1 Valid sequences of primitives

5.5.1.1 1.001

invoker invokes an operation receives result

5.5.1.2 1.002

invoker invokes an operation, expects error response

5.5.1.3 1.003

invoker invokes an operation receives failure (reason 1)

5.5.1.4 1.004

invoker invokes an operation receives failure (reason 2)

5.5.1.5 1.005

invoker invokes an operation, Doesn't receive any response (retransmission)

5.5.2 Invalid sequences of primitives

5.5.2.1 2.001

No sapbind

5.5.2.2 2.002

Invalid SAP

5.5.2.3 2.003

Invalid IP or no provider at dest

5.5.2.4 2.004

Invalid port no

5.5.2.5 2.005

Invalid op code

5.5.2.6 2.006

Invalid/unknown encoding type

5.5.3 Parameter variations on primitives

5.5.3.1 3.001

invoker invokes multiple operation, expects result

5.5.3.2 3.002

invoker invokes multiple operation, expects result and error

5.5.3.3 3.003

invoker invokes multiple operation, expects result and error

5.5.4 Stress Tests

5.5.4.1 4.001

invoker invokes operation with large data (100 bytes), expects result

5.5.4.2 4.002

invoker invokes operation with large data (1K bytes), expects result

5.5.4.3 4.003

invoker invokes 50 operations, expects 50 results

5.5.4.4 4.004

invoker invokes 100 operations, expects 100 results

5.5.4.5 4.005

invoker invokes 600 operations, expects 600 results

5.5.4.6 4.006

invoker invokes operation with 1K bytes data, expects result

5.5.4.7 4.007

invoker invokes operation with large data (2K bytes), expects result

5.5.4.8 4.008

invoker invokes 2000 operations, expects 2000 results

5.5.4.9 4.009

invoker invokes operation with 3K bytes data, expects result

5.5.5 Multiple Results

5.5.5.1 5.001

invoker invokes operation, receives multiple results for the same operation.

5.6 Example

The following pair of scenario files illustrates the structure and syntax of ESROS scenario files.

The test case is named 1.001. This numbering is simply a convention.

There is an invoker script and a performer script. The role of invoker or performer may be assumed by either the IUT or the LT.

The invoker issues, and the performer expects to see, an ESROS-INVOKE.request PDU containing the string ”date”. The performer responds with, and the invoker expects to see, an ESROS-RESULT.request PDU containing the string ”Feb 29, 1996”. The performer also checks for an ESROS-RESULT.confirm event.

```
log "--------------------------------"
log "inv1.001"
log "--- invoker invokes an operation"
log "--- receives result"
log "--------------------------------"
saprelease 12
sapbind 12 3
invoke request 13 2002 198.62.92.5 5 5 "date"
result indication 2 "Feb 29, 1996"
saprelease 12

log "--------------------------------"
log "perf1.001"
log "--- performer performs an operation"
log "--- receives result"
log "--------------------------------"
saprelease 13
sapbind 13 3 invoke indication 2 2 "date"
result request 2 "Feb 29, 1996"
result conformation
saprelease 13
```

Appendix A

Acronyms

ASN.1	Abstract Syntax Notation One (ASN.1)
FSM	ESROS Finite State Machine.
IP-Message	InterPersonal Message
ESROS	Efficient Short Remote Operation Services
ESROP	ESROS Protocol Engine
ESRO-SAP	ESROS Service Access Point.
MD	Management Domain
MH	Message Handling
MHS	Message Handling System
MS	Message Store
MT	Message Transfer
MTA	Message Transfer Agent
MTS	Message Transfer Service
SEQ_	Sequence Module
TMR_	Timer Management Module
TM_	Trace Module
DU_	Data Unit Management Module

Appendix B

ESRO API Example Usage

B.1 invoker.c

B.2 invoksch.c

B.3 performer.c

B.4 perfsch.c

Appendix C

ESRO Program man Pages

```
esros(1)                  User Commands                  esros(1)

NAME
     esros - Efficient Short Remote Operation Services Engine

SYNOPSIS
     esros [ -p port ] [ - s socket  ]  [   - o logfile  ]  [   -
     T module,mask ...  ]

DESCRIPTION
     esros implements the ESROS protocols  as  specified  in  LSM
     Protocol Specification Version 1.0.

OPTIONS
     -p port    This option sets the port  number  of  ESROS.  If
               this  option  is  not given, then the default port
               number is 2002.


     -s socket  This option applies  to  UNIX  version  of  esros
               only.  socket  is the name of socket used for com-
               munication between esros and application  program.
               The default value of socket is /tmp/SP.


     -o  logfile
               This option activates the PDU  logging  of  esros.
               The  incoming and outgoing PDUs are logged in log-
               file


     -T  module,mask
               This option activates the  tracing  capability  of
               esros  (if  it  is a version with built-in tracing
               feature). module is the name of a module that  can
```

be one of the following:

ESRO_ (User Shell), LOPS_ (Provider Shell), ESROP_ (Protocol Engine), FSM_ (Finite State Machine), IMQ_ (Inter Module Queue), SCH_ (scheduler), DU_ (Data Unit) or UDP_.

mask is a four digit hexadecimal number (0 to ffff) which controls different levels of trace output messages. ffff means a full trace output for the given module.

EXAMPLES

The following example:

```
example% esros -o out.log
```


enables PDU logging and writes the log of incoming and outgoing PDUs in the out.log file. The out.log format is binary and esropscop utility program is used to convert the log data to a readable format.

The next example activates the trace log of esros and generates the trace output of the FSM_ module (Finite State Machine):

```
example% esros -T FSM_,ffff
```

FILES

none

SEE ALSO

esropscop(1)

AUTHORS
     Neda Communications, Inc. -- Mohsen Banan, Kamran Ghane

REVISION
     RCS Revision: $Id: esros.1,v 1.1 1995/10/03 07:03:26 kamran
     Exp $

NAME
     esropscop - Display PDU log file created by esros

SYNOPSIS
     esropscop [-f follow] [-l line length] [-h] filename

DESCRIPTION

esropscop displays a readable format of lros log file on the standard output.

The column headers of the esropscop output have the following meanings:

TSDU	Transport Service Data Unit is a counter for all incoming and outging PDUs.
Tsiz	size of the PDU.
Loc	Local
Rem	Remote
Ref	Reference Number of the operation
Dst	Destination Service Access Point
Src	Source Service Access Point
OpVal	Operation Value
Encod	Encodingtype
Parameter	Parameter of the operation

OPTIONS

-f follow With the -f (follow) option, the program will not terminate after the line of the input-file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the log-file. Thus it may be used to monitor the growth of the log file that is being written by esros process.

-l line_length
This option sets the line length of the output. Each PDU is described on one line and the last column of the line is the PDU data. PDU data display is cut based on the line length. If this option is not given, the default value of line

length is 80.

-h When this option is given a complete display of PDU data in hexadecimal format is given for each PDU.

EXAMPLES

The following example:

```
example% esropscop -l 132 esros.log
```

displays the esros.log file and the maximum length of each line of output data is 132.

FILES

none

SEE ALSO

esros(1)

AUTHORS

Neda Communications, Inc. -- Mohsen Banan, Kamran Ghane

REVISION

RCS Revision: $Id: esropscop.1,v 1.1 1995/10/03 07:03:22 kamran Exp $

NAME

esrossi - Efficient Short Remote Operation Services Senario Interpreter

SYNOPSIS

esrossi [-T module,mask]

DESCRIPTION

esrossi interprets the scenario files and invokes operations. Invocations and expected results are defined in invoker scenario file. Expected invocation and their results are defined in performer scenario file.

OPTIONS

-T module,mask

This option activates the tracing capability of esros (if it is a version with built-in tracing feature). module is the name of a module that can be one of the following:

ESRO_ (User Shell), LOPS_ (Provider Shell), ESROP_ (Protocol Engine), FSM_ (Finite State Machine), IMQ_ (Inter Module Queue), SCH_ (scheduler), DU_ (Data Unit) or UDP_.

mask is a four digit hexadecimal number (0 to ffff) which controls different levels of trace output messages. ffff means a full trace output for the given module.

EXAMPLES

The following example:

```
example% esrossi -T ESRO_,ffff
```

enables trace output of the scenario interpreter.

FILES

none

SEE ALSO

esros(1)

AUTHORS

Neda Communications, Inc. -- Mohsen Banan, Kamran Ghane

REVISION

RCS Revision: $Id: esrossi.1,v 1.1 1995/10/03 07:03:30 kamran Exp $

Appendix D

Trace Bit Definitions

Module Name	**Trace Bit**	**Mask (hex)**	**Type of Tracing**
LOPS_	0	0001	TM_ENTER
		1	Set all levels of tracing
ESROP_	0	0001	TM_ENTER
	5	0020	TM_PDUIN
	6	0040	TM_PDUOUT
	10	0400	DU_MALLOC
		461	Set all levels of tracing
ESRO_	0	0001	TM_ENTER
	1	0002	ESRO_TRPRIM
		3	Set all levels of tracing
LOG_	0	0001	TM_ENTER
		1	Set all levels of tracing
ASN	0	0001	All ASN activity: formatting, parsing
		1	Set all levels of tracing
DU_	10	0400	All DU_ activity: allocate, free, link
		400	Set all levels of tracing
FSM_	2	0004	FSM_TMEXEC
	3	0008	FSM_TMGEN
	4	0010	FSM_TMFUNC
		1C	Set all levels of tracing
IMQ_	0	0001	All IMQ_ activity: allocate, free, link
		1	Set all levels of tracing
MM	1	0002	Error
	3	0004	Normal activity
	8	0100	PDU dump
		106	Set all levels of tracing
SCH_	0	0001	All Scheduler activity: queue manipulation, task execution
		1	Set all levels of tracing
UDP_	0	0001	All UDP_ activity: receive, bind
		1	Set all levels of tracing

Table D.1: Complete List of Trace Bit Definitions

Appendix E

Neda PICS for ESROS

E.1 Introduction

The following pages are the Neda Protocol Implementation Conformance Statement (PICS) which is based on the proposed PICS Proforma for ESROS [4]. All the tables appear here as they do in the Proforma, along with their corresponding numbers. Additional comments and explanations are provided where necessary.

E.2 Identification

E.2.1 Supplier Identification

The supplier information is to be provided in the following way:

Supplier Address	Neda Communications 17005 SE 31st Place Bellevue, WA 98008 USA
Contact Name	Mohsen Banan
Contact Address	Same as above
Phone Number	(425) 644-8026
Facsimile Number	(425) 562-9591
e-mail Address	sales@neda.com

E.2.2 Implementation Information

Implementation information shall be provided below

Name[1]	Neda's implementation of ESROS
Version Machine Configuration	1.06 Portable code Existing Platforms: Sun Sparc, Intel x86
Operating System	Portable code Existing Systems: Unix (Solaris), MS Windows

E.3 ESROS

E.3.1 ESROS Protocol Summary

Identification of the Protocol Specification	Efficient Short Remote Operations Protocol (ESROP)
Protocol Version(s) supported	1.03

E.3.2 ESROS Protocol Capabilities

E.3.2.1 Overview of ESRO Services

Item	Protocol Capability	Status	Reference	Support
ESROS1	Does the IUT support Acknowledged Result Service Mode?	O	1028-3.1.2	Yes:_x_ No:__ X:__
ESROS2	Does the IUT support Non-acknowledged Result Service Mode?	O	1028-3.1.3	Yes:_x_ No:__ X:__
ESROS3	Does the IUT support only the Serialized use of ESRO Services?	O	1028-3.1.4	Yes:__ No:_x_ X:__

Note on ESROS3: It is possible, based on compile time constants, to limit the support on only the serialized use of ESRO services.

E.3.2.2 Connectionless Oriented Operation

Item	Protocol Capability	Status	Reference	Support
LSRO-CL1	Does the IUT support Connectionless Oriented Operation?	O	1028-5.3.1	Yes:_x_ No:__ X:__
LSRO-CL2	Does the IUT support 3-Way Handshake?	O	1028-5.3.2	Yes:_x_ No:__ X:__
LSRO-CL3	Does the IUT support 2-Way Handshake?	O	1028-5.3.3	Yes:_x_ No:__ X:__

E.3.2.3 Segmentation and Reassembly

Item	Protocol Capability	Status	Reference	Support
ESROS-SR1	Does the IUT support Segmentation and Reassembly?	O	1028-5.3.4	Yes:__ No:_x_ X:__

E.3.2.4 Connection Oriented Operation

Item	Protocol Capability	Status	Reference	Support
ESROS-CO1	Does the IUT support Connection Oriented Operation?	O	1028-5.4.1	Yes:__ No:_x_ X:__

E.3.2.5 Concatenation and Separation

Item	Protocol Capability	Status	Reference	Support
ESROS-CS1	Does the IUT support Concatenation and Separation?	O	1028-5.6	Yes:__ No:_x_ X:__

Bibliography

[1] M. Banan, M. Taylor, and J. Cheng. AT&T/Neda's Efficient Short Remote Operations (ESRO) Protocol Specification Version 1.2. RFC 2188 (Informational), September 1997.

[2] Neda Public Document. *Open C Platform*. Neda Published Document 103-103-01, Neda Communications Inc, Bellevue, WA, October 1996. Online document is available at http://www.mailmeanywhere.org/sw.free/neda/foundations/ocp/OCP-MulPub/accessPage.html.