ESROS
Application Programming Interface

Neda Document Number: 103-101-06.03

Last Updated: 2000/03/23 21:01:43

Doc. Revision: 1.1.1.1

Neda Communications, Inc.

January 27, 1999

Contents

List of Tables

List of Figures

This document describes the Application Programming Interface for Efficient Short Remote Operation Services (ESROS).

Published by:
Neda Communications, Inc
17005 SE 31st Place
Bellevue, WA 98008

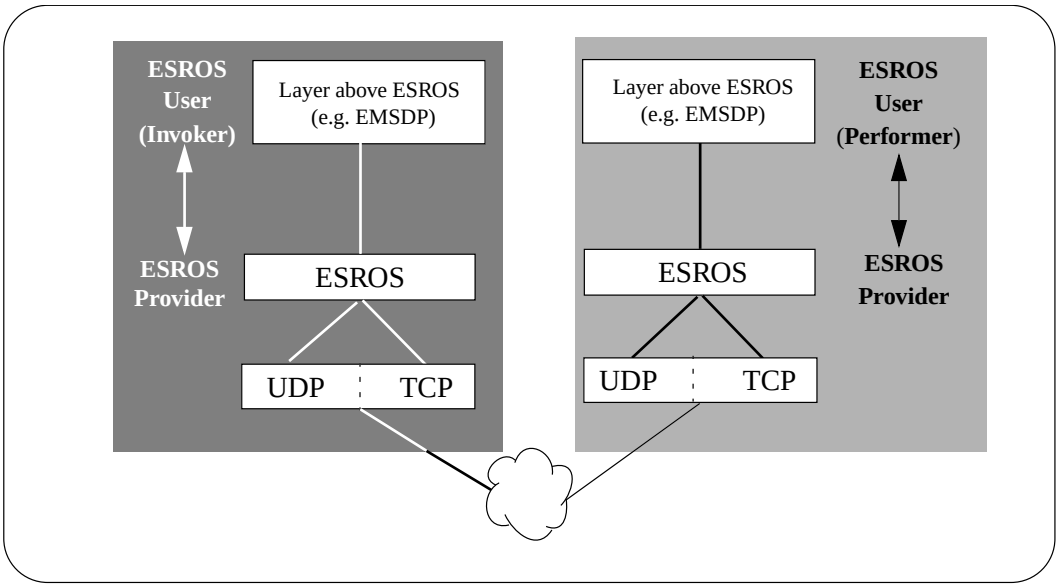

Figure 1: Implementation Architecture

1 Introduction

1.1 About This API

This document defines the ESROS' API. This definition conforms to RFC-2188 [2]. It is recommended that for this document to be of the most use to the reader, they should be familiar with RFC-2188[2] and Open C Platform [1].

Chapter 1 consists of an introduction to the API and the whole document.

Chapter 2 provides information about the interface to ESROS services.

Appendices include a Bibliography, a list of relevant Acronyms, ESROS API Example Usage, ESROS Program Man Pages.

1.2 Architecture

Figure 1, depicts the architecture of the complete ESRO protocols. ESROS-Daemon is responsible for implementation of ESRO-Protocol (RFC-2188. [2]) on both invoker and performer sides. ESROS-Daemon exposes the ESROS API (see chapter entitled ESRO API) to its users.

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.

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 1: ESRO Service Primitives

1.3 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.

Table 1 is a list of ESRO service primitive names.

The Neda ESROP upper interface conforms to the ESRO Service Definition [1]. The constraints on the sequence in which ESROP primitives may occur are explained in Reference [1].

1.3.1 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.

1.3.2 Operation Invocation

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

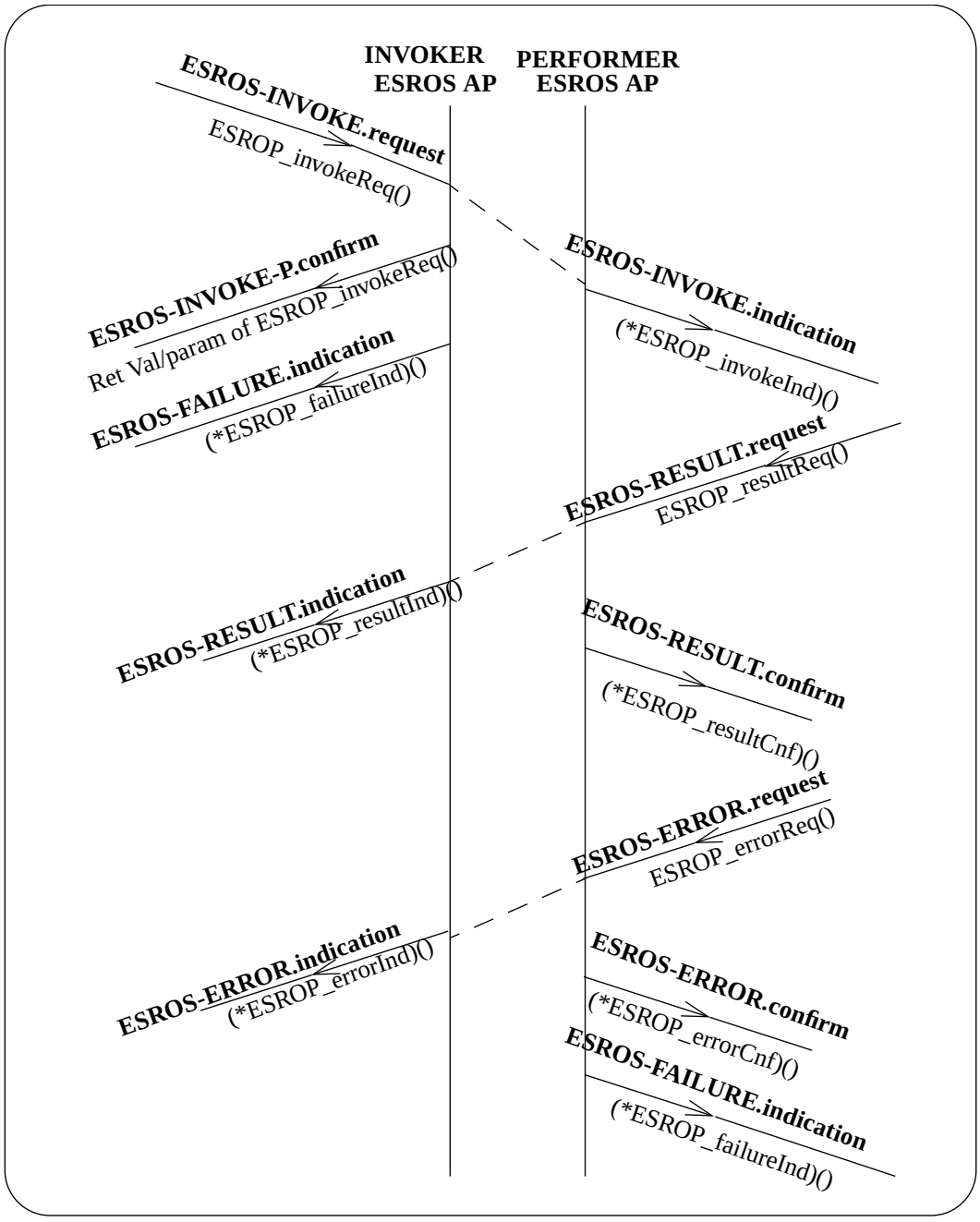

Figure 2: Time sequence diagram for ESRO Services

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: ESROS-SAP Management

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 3: Service Primitives and corresponding functions

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.

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 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.

1.4 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.

1.4.1 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.

1.4.2 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.

1.4.3 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.

1.4.4 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.

1.4.5 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.

1.4.6 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.

1.4.7 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:

/∗ *Events, Confirmation and Indications* ∗/

typedef struct ESRO_InvokeInd {
ESRO_SapDesc locSapDesc;
ESRO_SapSel remESROSap;
T_SapSel remTsap;
N_SapAddr remNsap;
ESRO_InvokeId invokeId;
ESRO_OperationValue operationValue;
ESRO_EncodingType encodingType;
Int len;
Byte data[ESRODATASIZE];
} ESRO_InvokeInd;

typedef struct ESRO_ResultInd {
ESRO_InvokeId invokeId;
ESRO_UserInvokeRef userInvokeRef,
ESRO_EncodingType encodingType;
Int len;
Byte data[ESRODATASIZE];
} ESRO_ResultInd;

```
typedef struct ESRO_ErrorInd {
ESRO_InvokeId     invokeId;
ESRO_UserInvokeRef          userInvokeRef,
ESRO_EncodingType           encodingType;
ESRO_ErrorValue  errorValue;
Int          len;
Byte         data[ESRODATASIZE];
} ESRO_ErrorInd;

typedef struct ESRO_ResultCnf {
ESRO_InvokeId     invokeId;
ESRO_UserInvokeRef          userInvokeRef,
} ESRO_ResultCnf;

typedef struct ESRO_ErrorCnf {
ESRO_InvokeId     invokeId;
ESRO_UserInvokeRef          userInvokeRef,
} ESRO_ErrorCnf;

typedef struct ESRO_FailureInd {
ESRO_InvokeId     invokeId;
ESRO_UserInvokeRef          userInvokeRef;
ESRO_FailureValue           failureValue;
} ESRO_FailureInd;

typedef struct ESRO_Event {
int type;                   /* Event type */
union {
          ESRO_InvokeInd   invokeInd;
          ESRO_ResultInd   resultInd;
          ESRO_ErrorInd    reorient;
          ESRO_ResultCnf   resultCnf;
          ESRO_ErrorCnf    errorCnf;
          ESRO_FailureInd  failureInd;
} un;
} ESRO_Event;


/* ESROS Event Primitive Codes */
#define ESRO_E_BASE      200
#define ESRO_INVOKE_IND  (ESRO_E_BASE+0)
#define ESRO_RESULT_IND  (ESRO_E_BASE+1)
#define ESRO_ERROR_IND   (ESRO_E_BASE+2)
#define ESRO_RESULT_CNF  (ESRO_E_BASE+3)
#define ESRO_ERROR_CNF   (ESRO_E_BASE+4)
#define ESRO_FAILURE_IND          (ESRO_E_BASE+5)
```

1.4.8 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.

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.

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.

1.5 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” [2]. 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

1.5.1 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.

1.5.2 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,
```

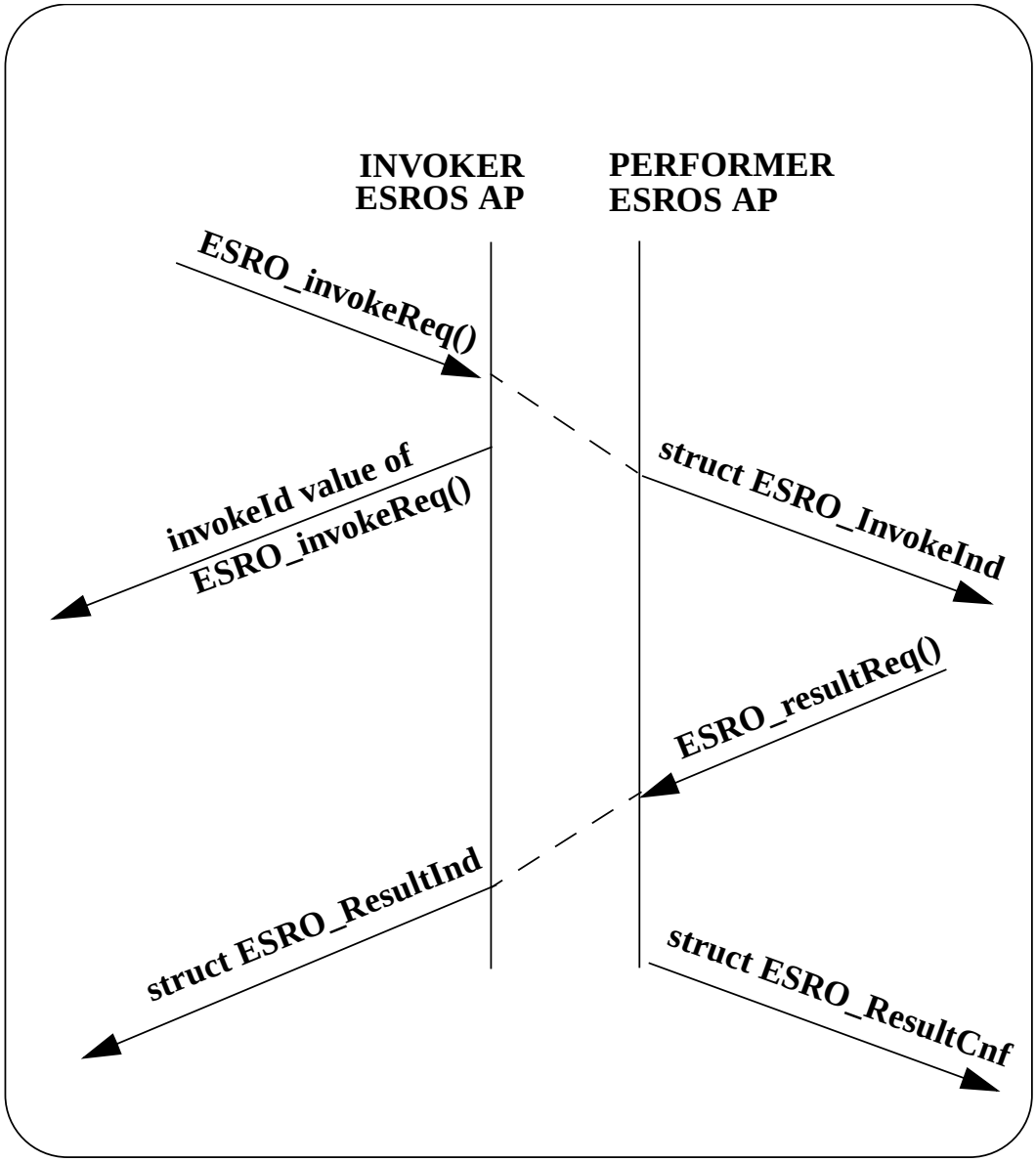

Figure 3: Example of time sequence diagram for ESROS Services

```
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),
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.

1.5.3 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.

1.5.4 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 [1] 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.

1.5.5 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.

1.5.6 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.

1.5.7 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.

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.

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.

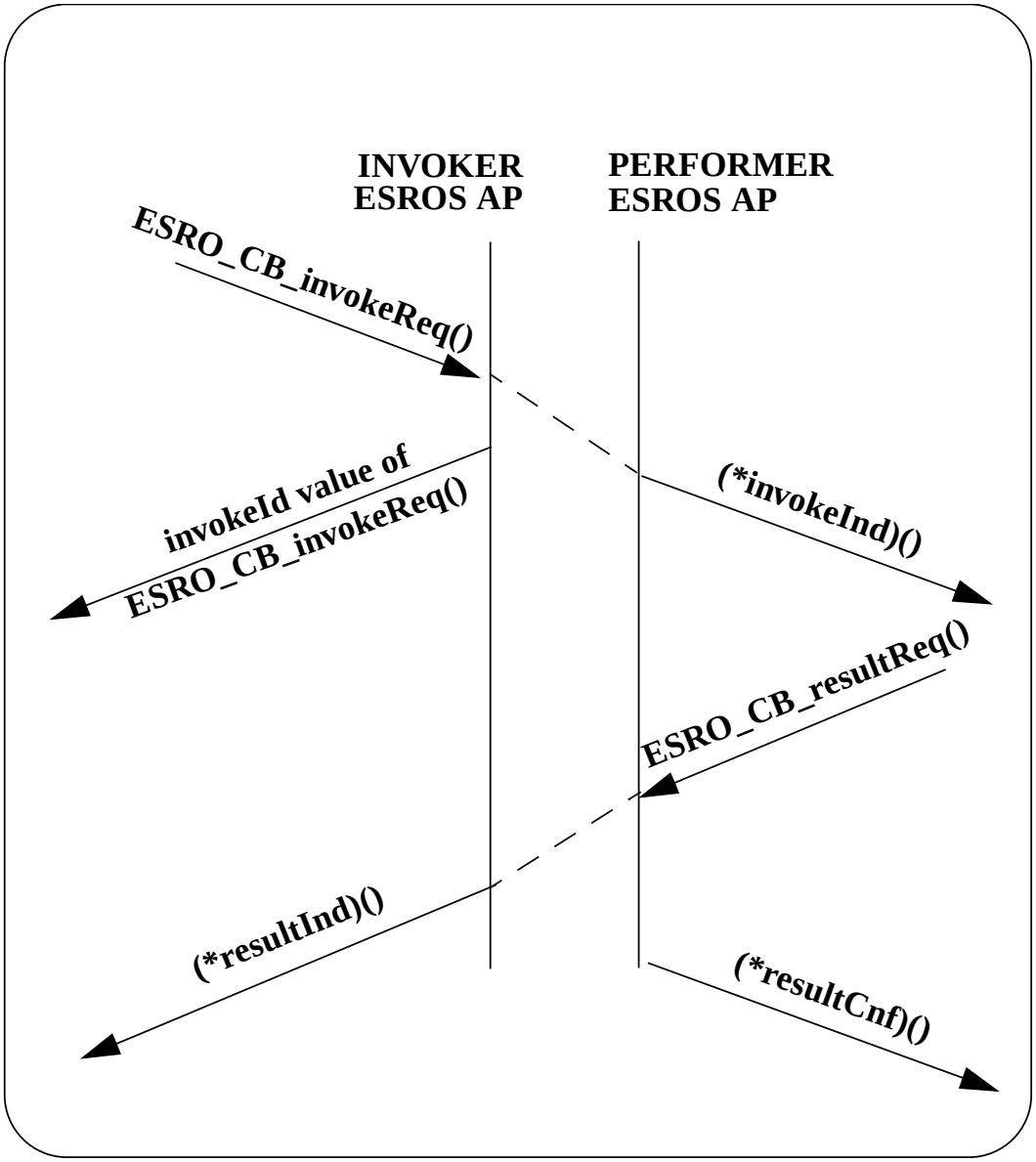

Figure 4: Example of time sequence diagram for ESROS CB Services

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

B ESRO API Example Usage

B.1 invoker.c

B.2 invoksch.c

```
/*+
 * File: invoksch.c
 *
 * Description: Invoker with call back ESROS API (with scheduler)
 *
 * Functions:
 *    G_heartBeat(void)
 *
-*/

#include "estd.h"
#include "pf.h"
#include "eh.h"
#include "du.h"
#include "getopt.h"
#include "tm.h"
#include "addr.h"
#include "inetaddr.h"
#include "sch.h"

#include "esro_cb.h"
#include "target.h"

#include "extfuncs.h"

RC_DEBUG_ENABLE;
char *__applicationName = "invokesch";

PUBLIC DU_Pool *G_duMainPool;

typedef struct G_Env {
    Char *progName;
    /* Application Specific Information */
} G_Env;

PUBLIC TM_ModDesc G_tmDesc;
PUBLIC G_Env G_env;

/* Quick temorary tracing */

Void G_init(void);
Void G_exit(Int code);
Void G_usage(void);
```

```
Void G_sigIntr(Int unused);
extern Int G_heartBeat(void);

ESRO_SapSel locSapSel = 14;
ESRO_SapDesc locSapDesc;

ESRO_InvokeId invokeId;
ESRO_EncodingType encodingType;

ESRO_OperationValue operationValue;
#define SHELL_CMD_OP  2

ESRO_SapSel remEsroSapSel = 13;
T_SapSel  remTsapSel  = {2, {0x22, 0x22}       }; /* UDP Port Number */
N_SapAddr remNsapAddr = {4, {198, 62, 92, 14} }; /* Remote IP Address */

ESRO_FunctionalUnit funcUnit = ESRO_3Way;

struct ESRO_Event event;

Bool reset = 0;

#ifdef TM_ENABLED
extern Void       DU_init();
#endif



/*<
 * Function:      G_init
 *
 * Description:  Initialize.
 *
 * Arguments:     None.
 *
 * Returns:       None.
 *
>*/

Void
G_init(void)
{
    G_tmDesc = TM_open("G_");
    if (!G_tmDesc) {
        EH_problem("G_init: TM_open G_ failed");
    }

    signal(SIGINT, G_sigIntr);
}
```

```
/*<
 * Function:       G_exit
 *
 * Description:  Exit.                                                    100
 *
 * Arguments:     Exit  code.
 *
 * Returns:        None.
 *
>*/

Void
G_exit(Int  code)
{                                                                         110
     exit(code);
}



/*<
 * Function:       G_usage
 *
 * Description:  Usage.
 *                                                                        120
 * Arguments:     None.
 *
 * Returns:        None.
 *
>*/

Void
G_usage(void)
{
     String  usage1 = "[-T G_,ffff]";                                     130
     String  usage2 = "-l localEsroSapSel -r remoteEsroSapSel -p remotePortNu -n remoteIPAdde";
     printf("%s: Usage: %s\n", G_env.progName, usage1);
     printf("%s: Usage: %s\n", G_env.progName, usage2);
}



/*<
 * Function:       G_sigIntr
 *                                                                        140
 * Description:  Signal  processing.
 *
 * Arguments:     None.
 *
```

```
 * Returns:      None.
 *
>*/

Void
G_sigIntr(Int unused)
{
/*
    ESRO_sapUnbind(locSapSel);          commented out to test ill-behaved invokers
*/
    signal(SIGINT, G_sigIntr);
    G_exit(22);
}



/*<
 * Function:     G_heartBeat
 *
 * Description: Heart Beat for the stack.
 *
 * Arguments:    None.
 *
 * Returns:      0 on successful completion, -1 on unsuccessful completion.
 *
>*/

SuccFail
G_heartBeat(void)
{
    if (SCH_block() < 0) {
        EH_fatal("main: SCH_block returned negative value");
        return (FAIL);
    }

    SCH_run();

    return (SUCCESS);
}

int
resultInd (ESRO_InvokeId invokeId,
           ESRO_EncodingType encodingType,
           DU_View parameter)
{
    char *duData;

    if (parameter) {
        duData = DU_data(parameter);
```

```
    } else {
        duData = "No parameter";
    }

    printf("------------------------------------------------------------------\n");
    printf("Invoker: Got Result Indication: invokeId=%d, paramter=%s\n",
            invokeId, duData);
    printf("------------------------------------------------------------------\n");

    reset = 1;

    return 1;
}


int
errorInd (ESRO_InvokeId invokeId,
            ESRO_EncodingType encodingType,
            ESRO_ErrorValue errorValue,
            DU_View parameter)
{
    char *duData;

    if (parameter) {
        duData = DU_data(parameter);

    } else {
        duData = "No parameter";
    }

    printf("------------------------------------------------------------------\n");
    printf("Invoker: Got Error Indication: invokeId=%d, paramter=%s\n",
            invokeId, duData);

    printf("------------------------------------------------------------------\n");

    reset = 1;

    return 1;
}

int
invokeIndication (ESRO_SapDesc locSapDesc, ESRO_SapSel remESROSap,
            T_SapSel *remTsap, N_SapAddr *remNsap, ESRO_InvokeId invokeId,
            ESRO_OperationValue opValue, ESRO_EncodingType encodingType,
            DU_View parameter)
{
    printf("\nInvoker: invokeIndication N/A\n");
    return 1;
}
```

```
int
resultCnf(ESRO_InvokeId invokeId)
{
    printf("\nInvoker: resultCnf N/A\n");
    return 1;
}

int
errorCnf(ESRO_InvokeId invokeId)
{
    printf("\nInvoker: errorCnf N/A\n");
    return 1;
}

int
failureInd(ESRO_InvokeId invokeId, ESRO_FailureValue failureValue)
{
    printf("\nInvoker: Failure Indication: InvokeId = %d \n", invokeId);
    return 1;
}



/*<
 * Function:      main()
 *
 * Description: main function of invoksch.
 *
 * Arguments:    argc, argv.
 *
 * Returns:       None.
 *
>*/

Int main(int argc, char **argv)
{
    Int c;
    Bool badUsage;

    G_env.progName = argv[0];
    TM_init();

    badUsage = FALSE;
    while ((c = getopt(argc, argv, "T:l:r:f:p:n:u")) != EOF) {
        switch (c) {
        case 'T':
            TM_setUp(optarg);
            break;
```

```
case 'l':       /* Local ESRO Sap Selector */
{
    Int gotVal;

    if ( PF_getInt(optarg, &gotVal, 12, 0, 63) ) {
        EH_problem("main: ");                                          300
        badUsage = TRUE;
    } else {
        locSapSel = gotVal;
    }
}
    break;

case 'r':       /* Remore ESRO Sap Selector */
{
    Int gotVal;                                                        310

    if ( PF_getInt(optarg, &gotVal, 13, 0, 63) ) {
        EH_problem("main: ");
        badUsage = TRUE;
    } else {
        remEsroSapSel = gotVal;
    }
}
    break;
                                                                       320
case 'p':       /* Remore Transport Sap Selector, UDP PortNu */
{
    Int portNu;

    if ( PF_getInt(optarg, &portNu, 0, 0, 10000) ) {
        EH_problem("main: ");
        badUsage = TRUE;
    } else {
        INET_portNuToTsapSel((MdInt) portNu, &remTsapSel);
    }                                                                  330
}
    break;

case 'n':       /* Remore NSAP Address, NSAP Address */
{
    struct in_addr inetAddr;

    * ((LgInt *) &inetAddr) = inet_addr(optarg);
    INET_in_addrToNsapAddr(&inetAddr, &remNsapAddr);
}                                                                      340
    break;

case 'f':       /* ESRO functional unit */
{
```

```
        Int gotVal;

        if ( PF_getInt(optarg, &gotVal, 12, 0, 63) ) {
            EH_problem("main: ");
            badUsage = TRUE;
        } else {                                                    350
            funcUnit = gotVal;
        }
    }
        break;

    case 'u':
    case '?':
    default:
        badUsage = TRUE;
        break;                                                      360
    }
}

while ((c = getopt(argc, argv, "T:u")) != EOF) {
    switch (c) {
    case 'T':
        TM_setUp(optarg);
        break;

    case 'u':                                                       370
    case '?':
    default:
        badUsage = TRUE;
        break;
    }
}

if (badUsage) {
    G_usage();
    G_exit(1);                                                      380
}

G_init();

G_duMainPool = DU_buildPool(MAXBFSZ, BUFFERS, VIEWS); /* build buf pool */

/* Application Specific Code Goes Here */
{
    String invokeParameter = "date";
    DU_View invokeDU;                                               390
    int paramLength;

    printf("\nRemote Network Address: %d.%d.%d.%d\n", remNsapAddr.addr[0],
            remNsapAddr.addr[1], remNsapAddr.addr,"RFC-2188"[2], remNsapAddr.addr,"RFC-2188"[3]);
```

```
    printf("\nFunctional Unit: %d-Way handshaking\n", funcUnit);

    /* Initialize All relevant modules */
    ESRO_init();

    TM_validate();                                                          400


    ESRO_CB_sapUnbind(locSapSel);

    if (ESRO_CB_sapBind(&locSapDesc, locSapSel,
                        funcUnit,
                        invokeIndication,
                        resultInd,
                        errorInd, resultCnf,
                        errorCnf, failureInd) < 0) {                        410
        EH_problem("invoke: Could not activate local ESRO SAP.");
        G_exit(13);
    }

    TM_TRACE((G_tmDesc, TM_ENTER,
             "invoker: Issue InvokeReq: remSapSel=%d, locSapDesc=%d, param=%s\n",
             remEsroSapSel, locSapDesc, invokeParameter));

    paramLength = strlen(invokeParameter);
    invokeDU = DU_alloc(G_duMainPool, paramLength);                         420
    OS_copy(DU_data(invokeDU), invokeParameter, paramLength);

    if (ESRO_CB_invokeReq(&invokeId, locSapDesc, remEsroSapSel,
                    &remTsapSel, &remNsapAddr,
                    (ESRO_OperationValue) SHELL_CMD_OP,
                    (ESRO_EncodingType) 2,  /* ASCII encoding */
                    invokeDU) < 0) {
        EH_problem("invoke: Could not Invoke");
        G_exit(11);
    }                                                                       430

    DU_free(invokeDU);

    while (!reset) {
        if (G_heartBeat()) {
            G_exit(13);
        }
    }


                                                                            440
    ESRO_CB_sapUnbind(locSapSel);

}
exit (SUCCESS);
```

```
} /* main() */
```

B.3 performer.c

```
/*+
 *
 * File: performer.c
 *
 * Description: Performer (ESROS), function call API.
 *
-*/

#ifdef OS_MALLOC_DEBUG
#include "os.h"

#undef FAIL
#endif

#include "estd.h"
#include "getopt.h"
#include "eh.h"
#include "tm.h"
#include "pf.h"

#include "esro.h"
#include "extfuncs.h"

RC_DEBUG_ENABLE;
char *__applicationName = "ops_xmpl performer";

extern char pubQuName[];

typedef struct G_Env {
    Char *progName;
    /* Application Specific Information */
} G_Env;

PUBLIC TM_ModDesc G_tmDesc;
PUBLIC G_Env G_env;

/* Quick temorary tracing */
#define G_tmHere()  TM_trace(G_tmDesc, TM_ENTER, "\n")

ESRO_SapSel locSapSel = 15;
ESRO_SapDesc locSapDesc;

ESRO_InvokeId invokeId;
ESRO_EncodingType encodingType;

ESRO_OperationValue operationValue;
```

```
#define SHELL_CMD_OP  2

struct ESRO_Event event;

/* Static functions */
static Int          perform(void);



/*<
 * Function:       main()
 *
 * Description: main function of performer.
 *
 * Arguments:     argc, argv.
 *
 * Returns:        None.
 *
>*/

Int main(int argc, char **argv)
{
    Int c;
    Bool badUsage;

    G_env.progName = argv[0];
    TM_init();

    badUsage = FALSE;
    while ((c = getopt(argc, argv, "T:l:s:")) != EOF) {
        switch (c) {
        case 'T':
            TM_setUp(optarg);
            break;

        case 'l':         /* Local ESRO Sap Selector */
        {
            Int gotVal;

            if ( PF_getInt(optarg, &gotVal, 13, 0, 63) ) {
                EH_problem("main (performer): ");
                badUsage = TRUE;
            } else {
                locSapSel = gotVal;
            }
        }
            break;

        case 's':
            if (strlen(optarg) != 0)
```

```
                strcpy(pubQuName, optarg);
            break;

        case 'u':                                                    100
        case '?':
        default:
            badUsage = TRUE;
            break;
        }
    }

    if (badUsage) {
        G_usage();
        G_exit(1);                                                   110
    }

    G_init();

    /* Do here what needs to be done */
    {
        ESRO_init();

        TM_validate();

                                                                     120
        perform();
    }

    exit (0);

} /* main() */




                                                                     130
/*<
 * Function:    G_init
 *
 * Description: Initialize.
 *
 * Arguments:   None.
 *
 * Returns:     None.
 *
>*/                                                                  140

Void
G_init(void)
{
    G_tmDesc = TM_open("G_");
    if (!G_tmDesc) {
```

```
        EH_problem("G_init : TM_open G_ failed");
    }

    signal(SIGINT, G_sigIntr);                                          150
}



/*<
 * Function:    G_exit
 *
 * Description: Exit.
 *
 * Arguments:   Exit code.                                              160
 *
 * Returns:     None.
 *
>*/

Void
G_exit(Int code)
{
    exit(code);
}                                                                       170



/*<
 * Function:    G_usage
 *
 * Description: Usage.
 *
 * Arguments:   None.
 *                                                                      180
 * Returns:     None.
 *
>*/

Void
G_usage(void)
{
    String usage1 = "";
    String usage2 = "";
    printf("%s: Usage: %s\n", G_env.progName, usage1);                  190
    printf("%s: Usage: %s\n", G_env.progName, usage2);
}
```

```
/*<
 * Function:    G_sigIntr
 *
 * Description: Signal processing.                                          200
 *
 * Arguments:   None.
 *
 * Returns:     None.
 *
>*/

Void
G_sigIntr(Int unUsed)
{                                                                           210
    signal(SIGINT, G_sigIntr);
    G_exit(22);
}

static void resultReq(String result);
/*
static void errorReq(String result);
*/


                                                                            220

/*<
 * Function:    perform()
 *
 * Description: Perform an operation.
 *
 * Arguments:   None.
 *
 * Returns:     None.
 *                                                                          230
>*/

static Int
perform(void)
{
    ESRO_RetVal gotVal;

    ESRO_sapUnbind(locSapSel);

    if ((gotVal = ESRO_sapBind(&locSapDesc, locSapSel, ESRO_3Way)) < 0) {   240
        EH_fatal("perform: Could not activate local ESRO SAP.");
    }


    while ( TRUE ) {
        while ((gotVal = ESRO_getEvent(locSapDesc, &event, 1)) < 0) {  /*????*/
```

```
            EH_problem("perform: ESRO_getEvent failed");
            return (FAIL);
        }
        if (gotVal < 0) {                                                    250
            EH_problem("perform: Bad Event");
        }
        processEvent(&event);
    }
}



/*<
 * Function:    processEvent                                                260
 *
 * Description: Process event.
 *
 * Arguments:   Event.
 *
 * Returns:     0 on successful completion, -1 on unsuccessful completion.
 *
>*/

SuccFail                                                                     270
processEvent(struct ESRO_Event *p_event)
{

    switch (p_event->type) {
    case ESRO_INVOKEIND:

        *(p_event->un.invokeInd.data + p_event->un.invokeInd.len) = '\0';

        TM_TRACE((G_tmDesc, TM_ENTER,
                  "Got ESRO-Invoke.Indcation invokeId=%d, operationValue=%d, paramter=%s\n",   280
                  p_event->un.invokeInd.invokeId,
                  p_event->un.invokeInd.operationValue,
                  p_event->un.invokeInd.data));

        /* Here is where you perform what you should do
         * and then call resultReq or errorReq.
         */
        invokeId = p_event->un.invokeInd.invokeId;

        {                                                                    290
            time_t idate;
            idate = time(&idate);
            resultReq(ctime(&idate));
        }
        break;
```

```
    case ESRO_RESULTCNF:

        TM_TRACE((G_tmDesc, TM_ENTER,
                  "Got ESRO-Result.Confirm invokeid=%d\n",                    300
                  p_event->un.resultCnf.invokeId));

        printf("-----------------------------------------------------\n");
        printf("Performer: Got Result Confirmation. invokeId=%d\n",
                  p_event->un.resultCnf.invokeId);
        printf("-----------------------------------------------------\n");
        G_exit(0);

    case ESRO_ERRORCNF:
        G_tmHere();                                                           310
        break;
    case ESRO_FAILUREIND:
        G_tmHere();
        break;
    default:
        G_tmHere();
        EH_problem("processEvent: invalid event type");
        break;
    }
    return (SUCCESS);                                                         320
}



/*<
 * Function:      resultReq
 *
 * Description: Result request.
 *                                                                           330
 * Arguments:     Result.
 *
 * Returns:       None.
 *
>*/

static void
resultReq(String result)
{
    ESRO_RetVal gotVal;                                                       340

    TM_TRACE((G_tmDesc, TM_ENTER, "Issuing ESRO-Result.Request parameter=%s\n",
              result));

    gotVal = ESRO_resultReq(invokeId, (ESRO_EncodingType) 2,
                            strlen(result), result);
```

```
    if (gotVal < 0) {
        EH_problem("resultReq: Could not Invoke");
    }

}
```

```
/*<
 * Function:       errorReq
 *
 * Description: Error request.
 *
 * Arguments:     Error.
 *
 * Returns:        None.
 *
>*/
/*
static void
errorReq(String result)
{
    ESRO_RetVal gotVal;

    gotVal = ESRO_errorReq(invokeId, (ESRO_EncodingType) 2,
                           strlen(result), result);

    if (gotVal < 0) {
        EH_problem("errorReq: Could not Invoke");
    }

}
*/
```

B.4 perfsch.c

```
/*+
 *
 * File: perfsch.c
 *
 * Description: Performer with call back ESROS API (with scheduler)
 *
 * Functions:
 *    G_heartBeat(void)
 *
-*/
```

```
#include "estd.h"
#include "pf.h"
#include "eh.h"
#include "du.h"
#include "getopt.h"
#include "tm.h"
#include "addr.h"
#include "inetaddr.h"
#include "sch.h"

#include "esro_cb.h"
#include "extfuncs.h"
#include "target.h"

RC_DEBUG_ENABLE;
char *__applicationName = "perfsch";

extern Int G_heartBeat(void);

PUBLIC DU_Pool *G_duMainPool;

typedef struct G_Env {
    Char *progName;
    /* Application Specific Information */
} G_Env;

PUBLIC TM_ModDesc G_tmDesc;
PUBLIC G_Env G_env;

ESRO_SapSel locSapSel = 15;
ESRO_SapDesc locSapDesc;

ESRO_InvokeId invokeId;
ESRO_EncodingType encodingType;

ESRO_OperationValue operationValue;
#define SHELL_CMD_OP  2

ESRO_FunctionalUnit funcUnit = ESRO_3Way;

struct ESRO_Event event;

Bool reset = 0;

#define RESULT  0
#define ERROR   1
Int resErr = RESULT;
```

```
/*<
 * Function:       G_init
 *
 * Description:  Initialize.
 *
 * Arguments:      None.
 *
 * Returns:        None.                                                    70
 *
>*/

Void
G_init(void)
{
     G_tmDesc = TM_open("G_");
     if (!G_tmDesc) {
          EH_problem("G_init : TM_open G_ failed");
     }                                                                      80

     signal(SIGINT, G_sigIntr);
}



/*<
 * Function:       G_exit
 *
 * Description:  Exit.                                                     90
 *
 * Arguments:      Exit code.
 *
 * Returns:        None.
 *
>*/

Void
G_exit(Int code)
{                                                                          100
     exit(code);
}



/*<
 * Function:       G_usage
 *
 * Description:  Usage.
 *                                                                        110
 * Arguments:      None.
 *
```

```
 * Returns:      None.
 *
>*/

Void
G_usage(void)
{
    String usage1 = "";                                                    120
    String usage2 = "";
    printf("%s: Usage: %s\n", G_env.progName, usage1);
    printf("%s: Usage: %s\n", G_env.progName, usage2);
}



/*<
 * Function:     G_sigIntr                                                130
 *
 * Description: Signal processing.
 *
 * Arguments:    None.
 *
 * Returns:      None.
 *
>*/

Void                                                                       140
G_sigIntr(Int unUsed)
{
    signal(SIGINT, G_sigIntr);
    G_exit(22);
}



/*<
 * Function:     G_heartBeat                                              150
 *
 * Description: Heart Beat for the stack.
 *
 * Arguments:    None.
 *
 * Returns:      0 on successful completion, -1 on unsuccessful completion.
 *
>*/

SuccFail                                                                   160
G_heartBeat(void)
{
```

```
    if (SCH_block() < 0) {
        EH_fatal("main: SCH_block returned negative value");
        return (FAIL);
    }

    SCH_run();

    return (SUCCESS);                                                    170
}


static void resultReq(String result);
static void errorReq(String error);

int
invokeIndication (ESRO_SapDesc locSapDesc, ESRO_SapSel remESROSap,
            T_SapSel *remTsap, N_SapAddr *remNsap, ESRO_InvokeId invokeId,
            ESRO_OperationValue opValue, ESRO_EncodingType encodingType,  180
            DU_View parameter)
{
        TM_TRACE((G_tmDesc, TM_ENTER,
                  "Got ESRO-Invoke.Indcation invokeId=%d, operationValue=%d, paramter=%s\n",
                  invokeId,
                  opValue,
                  DU_data(parameter)));

        /* Here is where you perform what you should do
         * and then call resultReq or errorReq.                           190
         */
        {
            time_t idate;
            idate = time(&idate);
            if (resErr == RESULT) {
                resultReq(ctime(&idate));
            } else {
                errorReq(ctime(&idate));
            }
        }                                                                  200

        return 1;
}




/*<
 * Function:      resultReq
 *                                                                        210
 * Description: Result request.
 *
```

```
 * Arguments:     Result.
 *
 * Returns:       None.
 *
>*/

static void
resultReq(String result)                                                    220
{
    ESRO_RetVal gotVal;
    DU_View resultDU;
    int paramLength;

    TM_TRACE((G_tmDesc, TM_ENTER, "Issuing ESRO-Result.Request parameter=%s\n",
            result));

    paramLength = strlen(result);
    resultDU = DU_alloc(G_duMainPool, paramLength);                         230
    OS_copy(DU_data(resultDU), result, paramLength);

    if ((gotVal = ESRO_CB_resultReq(invokeId, (ESRO_EncodingType) 2, resultDU))
        < 0) {
        EH_problem("resultReq: Could not Invoke");
    }

    DU_free(resultDU);

    reset = 1;                                                              240
}




/*<
 * Function:      errorReq
 *
 * Description: Error request.
 *                                                                          250
 * Arguments:     Error.
 *
 * Returns:       None.
 *
>*/

static void
errorReq(String error)
{
    ESRO_RetVal gotVal;                                                     260
    DU_View errorDU;
    int paramLength;
```

```
    TM_TRACE((G_tmDesc, TM_ENTER, "Issuing ESRO-Error.Request parameter=%s\n",
            error));

    paramLength = strlen(error);
    errorDU = DU_alloc(G_duMainPool, paramLength);
    OS_copy(DU_data(errorDU), error, paramLength);
                                                                                    270
    if ((gotVal = ESRO_CB_errorReq(invokeId, (ESRO_EncodingType) 2, 0, errorDU))
        < 0) {
        EH_problem("errorReq: Could not Invoke");
    }

    DU_free(errorDU);

    reset = 1;

}                                                                                   280

int
resultInd (ESRO_InvokeId invokeId,
           ESRO_EncodingType encodingType,
           DU_View parameter)
{
    char *duData;

    duData = DU_data(parameter);
                                                                                    290
        printf("-----------------------------------------------------------------\n");
        printf("Performer: Got Result Indication: invokeId=%d, paramter=%s\n",
                invokeId, duData);
        printf("-----------------------------------------------------------------\n");
    return 1;
}


int
errorInd (ESRO_InvokeId invokeId,                                                   300
           ESRO_EncodingType encodingType,
           ESRO_ErrorValue errorValue,
           DU_View parameter)
{
    char *duData;

    duData = DU_data(parameter);

        printf("-----------------------------------------------------------------\n");
        printf("Performer: Got Error Indication: invokeId=%d, paramter=%s\n",       310
                invokeId, duData);
```

```
        printf("-------------------------------------------------------------\n");

    reset = 1;

    return 1;
}

int
resultCnf(ESRO_InvokeId invokeId)
{
    TM_TRACE((G_tmDesc, TM_ENTER,
              "Got ESRO-Result.Confirm invokeid=%d\n",
              invokeId));

    printf("-----------------------------------------------------\n");
    printf("Performer: Got Result Confirmation. invokeId=%d\n",
           invokeId);
    printf("-----------------------------------------------------\n");

    reset = 1;

    return 1;
}

int
errorCnf(ESRO_InvokeId invokeId)
{
    TM_TRACE((G_tmDesc, TM_ENTER,
              "Got ESRO-Error.Confirm invokeid=%d\n",
              invokeId));

    printf("-----------------------------------------------------\n");
    printf("Performer: Got Error Confirmation. invokeId=%d\n",
           invokeId);
    printf("-----------------------------------------------------\n");

    reset = 1;

    return 1;
}

int
failureInd(ESRO_InvokeId invokeId, ESRO_FailureValue failureValue)
{
    printf("\nPerformer: Failure Indication: InvokeId = %d \n", invokeId);
    return 1;
}
```

```
/*<
 * Function:      main()
 *
 * Description: main function of performer.
 *
 * Arguments:    argc, argv.
 *
 * Returns:       None.                                                    370
 *
>*/

Int
main(int argc, char **argv)
{
    Int c;
    Bool badUsage;

    G_env.progName = argv[0];                                              380
    TM_init();

    badUsage = FALSE;
    while ((c = getopt(argc, argv, "T:l:f:re")) != EOF) {
        switch (c) {
        case 'T':
            TM_setUp(optarg);
            break;

        case 'l':       /* Local ESRO Sap Selector */                      390
        {
            Int gotVal;

            if ( PF_getInt(optarg, &gotVal, 13, 0, 63) ) {
                EH_problem("main (performer): ");
                badUsage = TRUE;
            } else {
                locSapSel = gotVal;
            }
        }                                                                  400
            break;

        case 'f':       /* ESRO functional unit */
        {
            Int gotVal;

            if ( PF_getInt(optarg, &gotVal, 12, 0, 63) ) {
                EH_problem("main: ");
                badUsage = TRUE;
            } else {                                                       410
                funcUnit = gotVal;
            }
```

```
    }
        break;
    case 'r':
        resErr = RESULT;
        break;
    case 'e':
        resErr = ERROR;
        break;
    case 'u':
    case '?':
    default:
        badUsage = TRUE;
        break;
    }
}

if (badUsage) {
    G_usage();
    G_exit(1);
}

G_init();

G_duMainPool = DU_buildPool(MAXBFSZ, BUFFERS, VIEWS); /* build buf pool */

ESRO_init();

TM_validate();

printf("\nFunctional Unit: %d-Way handshaking\n", funcUnit);

ESRO_CB_sapUnbind(locSapSel);

if (ESRO_CB_sapBind(&locSapDesc, locSapSel,
                    funcUnit,
                    invokeIndication,
                    resultInd,
                    errorInd, resultCnf,
                    errorCnf, failureInd) < 0) {
    EH_fatal("perform: Could not activate local ESRO SAP.");
}

while (!reset) {
    if (G_heartBeat()) {
        G_exit(13);
    }
}

ESRO_CB_sapUnbind(locSapSel);
```

```
    exit (0);

} /* main() */
```

References

[1] 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.public.neda.com/pubs/biblio/103-103-01/index.html.

[2] M. Taylor, J. Cheng, and M. Banan. *AT&T/Neda's Efficient Short Remote Operations (ESRO) Protocol Specification Version 1.2.* Request for Comments (Informational) 2188, Neda Communications, Inc., September 1997. Online document is available at ftp://ftp.isi.edu/in-notes/rfc2188.txt.