Neda's Implementation of EMSD-UA – Source

Neda Document Number: 105-101-08

Last Updated: Author unspecified

Doc. Revision: source unspecified

Neda Communications, Inc.

1996

Preface

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

Neda Communications, Inc.
Phone: +1 425 644-8026
Fax: +1 1 425 562-9591
E-Mail: info@neda.com

Contents

Chapter 1

Introduction

1.1 About This Document

This document is available in PDF, Postscript, and HTML format.

1.2 Overview of Architecture

Figure 1.1 illustrate the high level software architecture for this product (EMSD-UA-Source)

```
+---------+     +---------+     +---------+
|         |     |         |     |         |
|pine-lsm |     |pager-emu|     |combined |
|         |     |         |     |         |
+---------+     +---------+     +---------+
     ^               ^               ^
     |               |               |
     +-------------+ | +-------------+
                   | | |
                   v v v
                +---------+
                |         |
                | UA-CORE |
                |         |
                +---------+
                     ^
                     |
                     |
                     |
                     v
                +---------+
                |         |
                |  ESROS  |
                |         |
                +---------+
```

Figure 1.1: Overview of Architecture

Chapter 2

EMSD-UA Architecture

2.1 Overview

The EMSD User Agent is composed of a number of cooperating modules. Some modules cooperate by one module calling functions in another module as needed. Other modules cooperate via a call-back methodology where functions in one module are registered with another module, so that when a particular event or action occurs, the registered function will be called.

Figure 2.1 shows the relationship of the primary modules associated with the UA. Note that the modules with which the UA directly cooperates may, themselves, interact with other modules, but those other modules and interactions are not necessarily depicted here.

The User Agent Core implements the EMSD protocol, and maintains the state machines required for the operation thereof. The User Agent Shell implements the specifics of how messages retrieved from, and provided to, the user interface (i.e. it implements its own user interface, obtains spooled files created by a Pine UA, etc.).

2.2 The User Agent Core

The User Agent Core uses, as its module descriptor, ”UA”. Per the Neda coding standards, functions which are intended to be called from outside of the module use an uppercase module descriptor. The first function to be called is UA_init(), found in file ”uacore/ua.c”. UA_init() receives many parameters. The documentation in ”ua.h”

```
+-----------------------------------------------------------------------+
|                                                                       |
| v and ^ = function call interface                                     |
|                                                                       |
| v     . = call-back interface through registered functions            |
| . and ^                                                               |
|                                                                       |
|    +---------------------------------------+              +---------+ |
|    |                                       |              |         | |
|    |                                       |              |         | |
|    |             User Agent Shell          |>>>>>>>>>>>>>>|  SCH    | |
|    |                                       |              |         | |
|    |                                       |              |         | |
|    +---------------------------------------+              +---------+ |
|       v                     .                                         |
|       v                     ^                             +---------+ |
|       v                     .                             |         | |
|       v Submission          ^ Delivery                    |         | |
|       v and other           . and other             >>>>>>|  ASN    | |
|       v requests            ^ events               ^      |         | |
|       v                     .                      ^      |         | |
|       v                     ^                      ^      +---------+ |
|       v                     .                      ^                  |
|    +---------------------------------------+       ^      +---------+ |
|    |                                       |>>>>>>>^      |         | |
|    |                                       |              |         | |
|    |             User Agent Core           |>>>>>>>>>>>>>>|  FSM    | |
|    |                                       |              |         | |
|    |                                       |>>>>>>>v      |         | |
|    +---------------------------------------+       v      +---------+ |
|       v                     ^                      v                  |
|       v                     .                      v      +---------+ |
|       v                     ^                      v      |         | |
|       v                     .                      v      |         | |
|       v                     ^                       >>>>>>|  NVM    | |
|       v InvokeRequest,      . InvokeIndication,           |         | |
|       v ResultRequest       ^ ResultIndication,           |         | |
|       v etc.                . ResultConfirm               +---------+ |
|       v                     ^ etc.                                    |
|       v                     .                                         |
|       v                     ^                                         |
|       v                     .                                         |
|       v                     ^                                         |
|    +---------------------------------------+                          |
|    |                                       |                          |
|    |                                       |                          |
|    |                ESROS                  |                          |
|    |                                       |                          |
|    +---------------------------------------+                          |
|                                                                       |
+-----------------------------------------------------------------------+
```

Figure 2.1: Relationship of Primary UA Modules

describes each of them, so they won't be explicitly explained here.

In the course of processing the UA Core initialization code, the following significant actions take place:

- The pointers to the call-back functions are saved
- The password string is copied so that the caller need not maintain a permanent copy of it
- A trace handle for the UA Core module is generated
- The UA Core uses non-volatile memory to maintain both Submission Verify information and Duplicate Detection information. At initialization time, the non-volatile memory is examined to see if it has been set up properly to maintain these information types. If not, then appropriate setup is accomplished, involving creating NVM transactions, allocating memory as part of the transaction, initializing portions of the allocated memory including queue pointers, and ending the NVM transactions.
- ASN-compile the parse trees for each of the EMSD protocol elements.
- Bind to ESROS Service Access Points so that messages can be sent and received via the EMSD protocol. Binding the ESROS SAP's also provides pointers to call-back functions within the UA Core, which ESROS calls when Indications and Confirmations are received.
- Initialize the finite state machines used to implement the EMSD protocol, allowing multiple concurrent operations.

2.2.1 To process or Preprocess, that is the question

Preprocessing is done on all events incoming to the UA Core module, regardless of whether those events originate at the UA Shell level, or at the ESROS level. The type of preprocessing, however, differs depending on the event being processed. The preprocessing functions are found in file ”uacore/upreproc.c”.

Preprocessing of events from ESROS involve parsing the incoming PDU, and either associating the event with an existing operation, or allocating a new operation structure for the operation. Also associated with each operation is a finite state machine, a pointer to which is maintained in the operation structure.

Submission requests require a slightly different type of preprocessing. With this event type, the message is converted to the internal format required of the ASN formatter, a PDU is formatted, and an operation and finite state machine are allocated.

2.2.2 Submission

When the UA Shell has messages for submission, it calls the function UA_submitMessage(), in file ”uacore/ua.c”. This function preprocesses the request and then generates an event to the finite state machine to process the request. The finite state machine handles the remainder of the message submission. See the section entitled ”Finite State Machines”. Once the event has been generated to the finite state machine, the submission function returns to its UA Shell caller. The finite state machine is run via the SCH scheduler module, from the main loop in the UA Shell main() function.

2.2.3 Delivery

Events from ESROS invoke call-back functions (registered by UA_init()) which are located in the file ”uacore/lsroevt.c”. When an INVOKE.indication arrives from ESROS, the function ua_invokeIndication() is called. The first step of processing the INVOKE.indication is to determine whether the invoking operation is a duplicate of a previous operation. The mechanism of this determination is described in the section ”Duplicate Detection”. If this operation is a duplicate, a RESULT.request is sent back, just as if the operation had actually been accomplished – since, in reality, the operation was accomplished by the original PDU for which this is a duplicate. The PDU is then discarded, and no further processing occurs.

Assuming that the PDU is not a duplicate, it is preprocessed (see above). Since the preprocessing provides an operation structure and a finite state machine for this operation, an event can now be generated to the finite state machine, for further processing of this operation. Once the event has been generated to the finite state machine, the call-back function returns to its ESROS caller. The finite state machine is run via the SCH scheduler module, from the main loop in the UA Shell main() function.

2.2.4 Duplicate Detection

The code to implement duplicate detection is found in the file ”uacore/uadup.c”.

When a DELIVER.request is to be sent, the function ua_addDuplicateDetection() is called, to add a duplicate detection octet. The value of this octet is simply incremented by one with each successive call.

Whenever a PDU is received from ESROS, it is checked to determine if the requested operation is a duplicate of a prior request. If it is, the request must be ignored. The function ua_isduplicate() is called to accomplish this checking.

2.2.5 Finite State Machines

The finite state machines for the UA performer are found in the file ”uacore/uperform.c”. The finite state machine is pictorially depicted at the top of this file, and an understanding of this graphic is essential to comprehension of the workings of the FSM.

The finite state machines for the UA invoker are found in the file ”uacore/submit.c”. Again, the finite state machine is pictorially depicted at the top of this file, and an understanding of this graphic is essential to comprehension of the workings of the FSM.

Function names within these files are chosen to indicate clearly the purpose and time-of-use of the functions. Most of the functions are of two types: Predicate or Action. A predicate is used to determine whether a particular transition between states is to occur. Once a predicate function returns TRUE, the associated action function is called to process the state transition. Predicates and Actions are associated with the state in which an event occurs, and with the resulting state following the transition. In order to allow for multiple possible transitions between the same two states, a single letter suffix is used.

As an example of function naming, take the transition, in ”uacore/submit.c”, from the the Invoke Response Wait state, State 2, to the Idle state, State 1, as a result of receiving a RESULT indication. This is shown in the graphic as an arrow going from the State 2 box to the State 1 box, with the designator [A]. The predicate, for determining if this transition is to be taken, is therefore called pS2S1a():

p = Predicate S2 = Starting state is S2 S1 = Ending state is S1 a = Of the possible transitions from S2 to S1, this is [A]

Note that if there is only one possible transition from a given state, for a particular event, a predicate function is not required. Predicate functions are only required when the same event could result in two different possible transitions, and it is the job of the predicate function to determine whether a transition should be taken.

Processing of events in the finite state machine occur through scheduling accomplished by the SCH scheduler module. When an event is generated to the finite state machine, by a call to FSM_generateEvent(), the finite state machine is placed on the scheduler queue to be run. When the scheduler runs this task, the predicate and action functions appropriate to the event and operation data are processed. (In some cases, this means that the registered call-back functions in the UA Shell are called.)

2.3 The User Agent Shell

It is recommended that the reader refer to ”mts_ua/include/ua.h” for a detailed description of the interface that the User Agent Core module requires of the User Agent Shell. In particular, the documentation of the UA_init() function describes each of the callback functions which the UA Shell must implement, and the parameters of each; and the documentation of UA_submitMessage() which the UA Shell must call to submit a message to the UA Core. From this information, it should be obvious what a User Agent Shell must do to send and receive messages using the facilities of the User Agent Core.

Appendix A

Bibliography

[103-100-14] OPEN C Platform, 1995 Neda Communications, Inc.

[4] Neda Communications, Inc., ”Limited Size Remote Operations: Protocol Implementation Conformance Statement”.

Banan, et.al., ”Efficient Short Remote Operations”, RFC 2188, Neda Communications Inc, September 1997.

Appendix B

Trace Bit Definitions

Module Name	Trace Bit	Mask (hex)	Type of Tracing
???	0	0001	????
		1	Set all level of tracing
???	0	0001	
	5	0020	
	6	0040	
	10	0400	
		461	

Table B.1: Complete List of Trace Bit Definitions