TRILLIUM®

PSF - ISUP (FT/HA)

Training Course

1095146 1.2

PSF - ISUP (FT/HA)

Training Course
1095146 1.2

Trillium Digital Systems, Inc.
12100 Wilshire Blvd., Suite 1800
Los Angeles, CA 90025-7118
Phone: +1 (310) 442-9222
Fax: +1 (310) 442-1162
Web: http://www.trillium.com

PSF - ISUP (FT/HA)
Training Course
1095146 1.2

Information furnished herein by Trillium Digital Systems, Inc., is believed to be accurate and reliable. However, Trillium assumes no liability for errors that may appear in this document, or for liability otherwise arising from the application or use of any such information or for any infringement of patents or other intellectual property rights owned by third parties which may result from such application or use. The products, their specifications, and the information appearing in this document are subject to change without notice.

To the extent this document contains information related to software products you have not licensed from Trillium, you may only apply or use such information to evaluate the future licensing of those products from Trillium. You should determine whether or not the information contained herein relates to products licensed by you from Trillium prior to any application or use.

Printed in U.S.A.

Preface

Objective

This document forms the training material for the PSF - ISUP (FT/HA) software (p/n 1000146) designed by Trillium Digital Systems, Inc.

Audience

Trillium assumes that the readers of this document are familiar with telecommunication protocols, specifically SS7 and Trillium's ISUP, PSIF - ISUP, and Fault-Tolerant/High Availability Core products.

Document Organization

This document is organized into the following sections:

Section	Description
1 Deliverables	Lists the media, documentation, and support provided for the PSF - ISUP (FT/HA) software
2 Software Organization	Describes the environment, architecture, and the primitives of PSF - ISUP (FT/HA)
3 Files	Lists all the product, common, and sample files. A diagram illustrates how the files are organized.
4 Functional Description	Describes all the features of PSF - ISUP (FT/HA)
5 Interfaces	Defines the systems services, layer manager, upper layer, and lower layer interfaces
6 Internal Organization	Describes the Service Access Points (SAPs), user connections, state transition matrices, message database, and timer management
7 Porting	Lists the porting sequence

Document Set

The suggested reading order of this document set is:

1. *Functional Specification*

   Contains the features and highlights that describe the protocol and system characteristics. It includes the memory characteristics and conformance details.

2. *Training Course*

   Offers a detailed overview of the features and interfaces of the software. It contains code samples, data flow diagrams, and a list of files.

3. *Service Definition*

   Describes the procedures and layer manager interface used to pass information between the software and other software elements. The Interface Primitives section describes the services of the software. The Interface Procedures section describes and illustrates the flow of primitives and messages across the interfaces.

   Note: *Information on porting the software is contained in the Service Definition.*

4. *Software Test Sample*

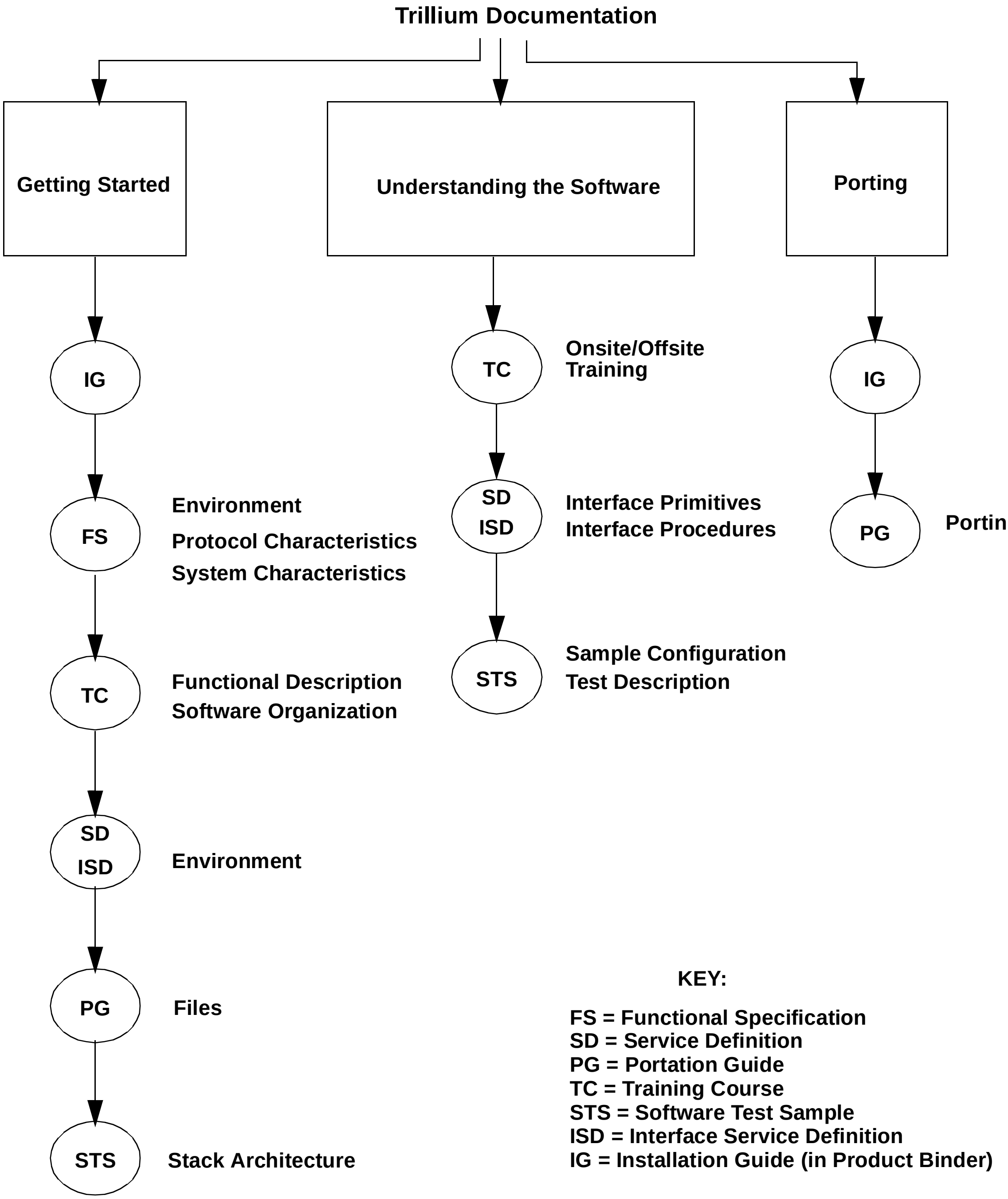
   Describes the sample files delivered with the product and the procedures to build a sample test. This test partially demonstrates the product initialization, configuration, and execution. It may contain data flow diagrams illustrating the correct operation of the software.

In addition to the above PSF documents, the following documents should also be read for a better understanding of the fault-tolerant system:

1. *Fault-Tolerant/High-Availability (FT/HA) Core Functional Specification*
2. *Fault-Tolerant/High-Availability (FT/HA) Core Service Definition*

Using Trillium Documentation

The figure below illustrates the various approaches the user can take when utilizing the software documentation. First time users should read the documents under the **Getting Started** column; important sections and subsections are listed to the right of each document. For users familiar with the documentation but who need to look up certain points concerning the use of the software, the **Understanding the Software** column is suggested. The **Porting** column is for those users who are familiar with Trillium software and related telecommunications protocols and who wish to install the software immediately onto their operating systems.



Notations

This table displays the notations used in this document:

Notation	**Explanation**	**Examples**
Arial	**Titles**	**1.1 Title**
Palatino	Body text	This is body text.
Bold	**Highlights information**	**Loose coupling, tight coupling**, **upper layer interface**
ALL CAPS	CONDITIONS, MESSAGES	AND, OR CONNECT ACK
Italics	*Document names, emphasis*	*PSF - ISUP (FT/HA) Training Course* This adds *emphasis*.

`Courier New Bold`	**`Code`**
	`Filenames, pathnames`
		`PUBLIC S16 ZiMiLziCfgReq(pst, cfg)`
		`Pst      *pst;`
		`CmPFthaMngmt  *cfg;`

Release History

This table lists the history of changes in successive revisions to this document:

Version	**Date**	**Initials**	**Description**
1.2	December 31, 1999	sk	Changes for software release 1.2, including: • Addition of multiple point code support • Addition of NTT and Bellcore variants • FT/HA support for PSIF - ISUP
1.1	November 16, 1998	rs	• Initial release for software version 1.1

Contents

Illustrations

1 DELIVERABLES

- Software
- Documentation
- Support

Deliverables

Software

- Source (**`.c,.x,.h`**) and command files (**`.mak`**) that can be used to generate the relocatable object of the software product
- Written in ANSI and K&R C
- Delivered on CD-ROM

Deliverables

Documentation

- *Functional Specification*

  Describes the protocol, system, and performance characteristics of the software

- *Service Definition*

  Describes the data and procedures used to pass information between the PSF - ISUP software and the other software elements with which it interacts. These elements include PSF - ISUP, peer PSF - ISUP, system services, and the layer manager. This document also serves as the *Portation Guide* for the PSF - ISUP software.

- *Software Test Sample*

  Describes the sample files delivered with the product and the procedures needed to build a sample executable. The tests in the *PSF - ISUP Software Test Sample* demonstrate initialization, configuration, and execution of the product.

- Delivered in Adobe PDF

- Delivered as printed manuals and on CD-ROM

Deliverables

Support

- Warranty

  Product updates, technical bulletins, and documentation updates as published

  Telephone and e-mail assistance for correction of problems

- Maintenance

  Product updates, technical bulletins, and documentation updates as published

  Telephone and e-mail assistance for correction of problems

Deliverables

Support

Software Problem Reports

- Check your documentation

- Gather all the information that applies to your problem:
  - Enable error checking (**`ERRCLASS`**)
  - Enable trace, if needed (**`TRCx`**)
  - Note system error number, if any (**`SLogError`**)
  - Enable debug prints, if needed
  - Collect protocol information (such as messages, elements, and cause / diagnostic codes)

- Contact Trillium Technical Support at:

Email	**Telephone**	**Fax**
tech_support@trillium.com	+1 310 442 9222	+1 310 442 1162

2 SOFTWARE ORGANIZATION

- Environment

- Trillium Advanced Portability Architecture (TAPA)
  - Standard protocol layer
  - PSF - ISUP

- Primitives

- PSF - ISUP stack architecture

- Definitions

Software Organization

Environment

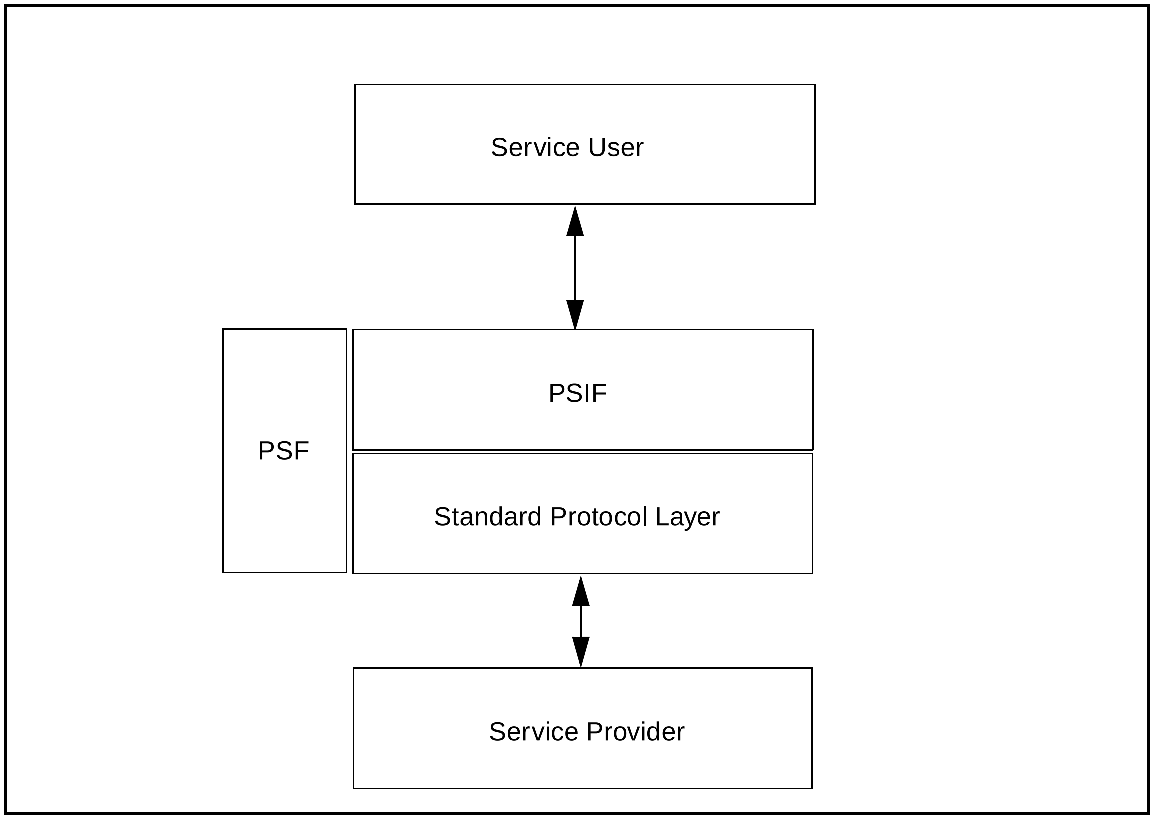


Figure 2-1: Environment

Software Organization

Environment

- Typically, each Trillium software product implements a standard protocol layer. However, PSF - ISUP software is not a standard product layer. PSF - ISUP is designed to provide Fault-Tolerant / High-Availability (FT / HA) capability to Trillium's ISUP and PSIF - ISUP products.

- Each Trillium software product consists of a self-contained set of functions that provide the features described in the appropriate *Functional Specification*. PSF - ISUP software is based on the *Fault-Tolerant/High-Availability (FT/HA) Functional Specification* defined by Trillium.

- The software has been designed so that the service user does not need to be aware of the implementation details of the service provider.

- The software is invoked by a small set of primitives. Primitives are a set of C function calls and are described in the appropriate *Service Definition*.

Software Organization

Trillium Advanced Portability Architecture

Standard Protocol Layer

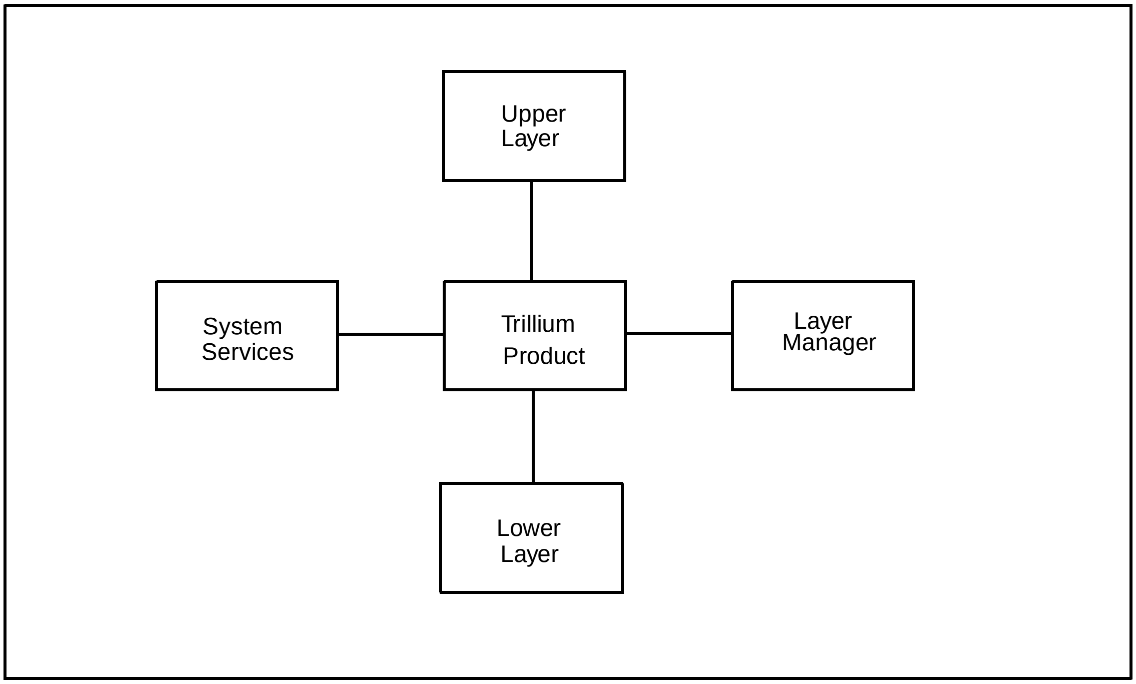


Figure 2-2: TAPA (standard protocol layer)

- The interfaces with upper layer, lower layer, and layer manager can each be tightly coupled (direct function call interface) or loosely coupled (message passing interface). System services is always tightly coupled.

Software Organization

Trillium Advanced Portability Architecture

PSF - ISUP

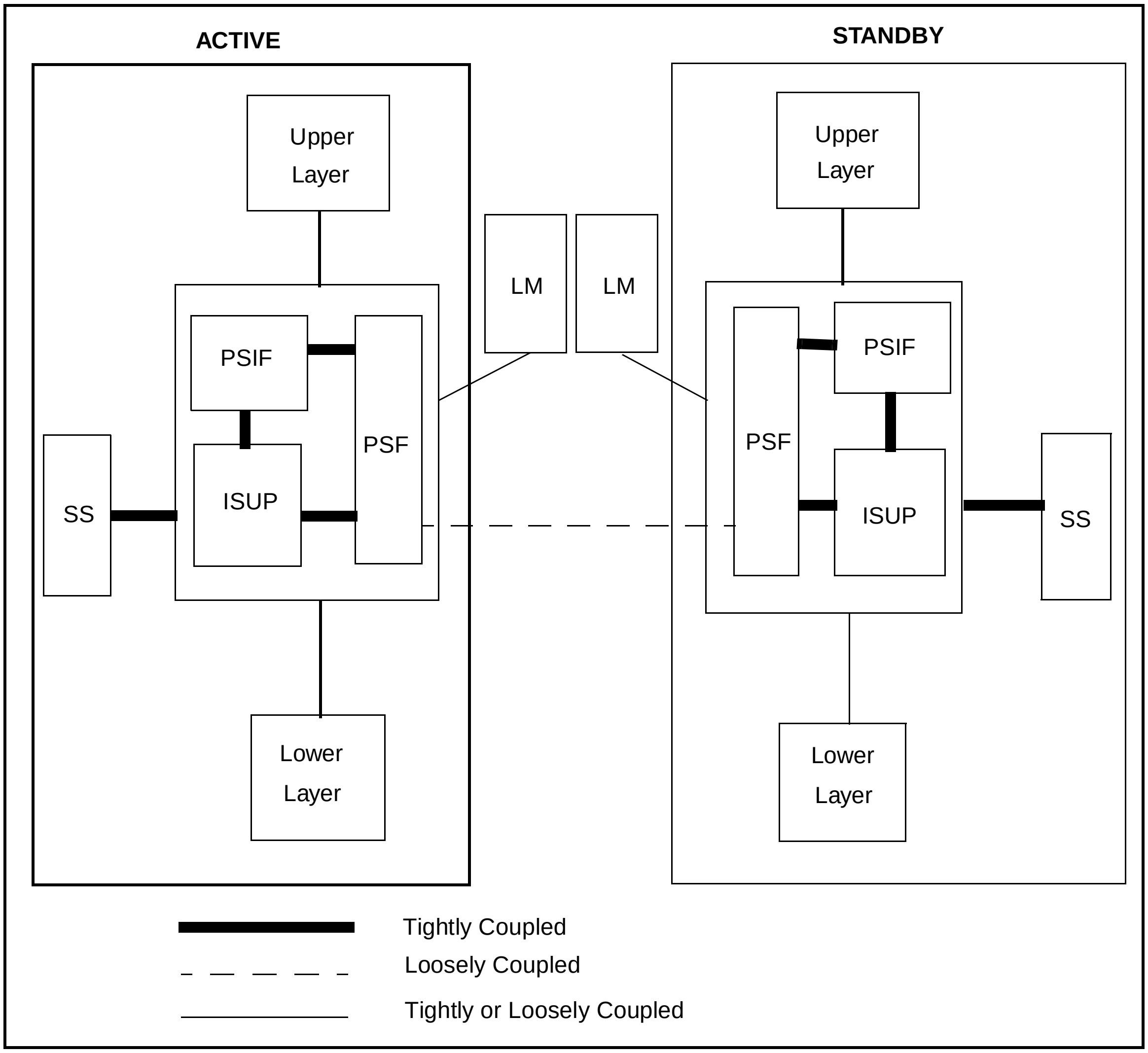


Figure 2-3: TAPA: PSF - ISUP

Software Organization

Trillium Advanced Portability Architecture

PSF - ISUP

- The system services interface provides the system functions required by PSF - ISUP, including, initialization, timer management, memory management, message and queue management, date and time management, and resource checking. The System Services interface is always tightly coupled.

- The layer manager interface provides the functions required to control and monitor PSF - ISUP. In addition, this interface provides the functions to initialize and modify PSF - ISUP configuration parameters. This interface can be tightly or loosely coupled.

- The peer PSF - ISUP interface provides functions for the active PSF - ISUP to communicate with the standby PSF - ISUP, and vice versa. These functions involve state updates from active to standby during run time, warmstart, and controlled switchover. This interface is always loosely coupled.

- The PSF - ISUP and ISUP (PSIF - ISUP) interface is always tightly coupled. PSF - ISUP updates the stable state changes during run-time state update. In order to avoid excessive overhead, all transient states are updated during warmstart and controlled switchover, but not during run time.

Software Organization

Primitives

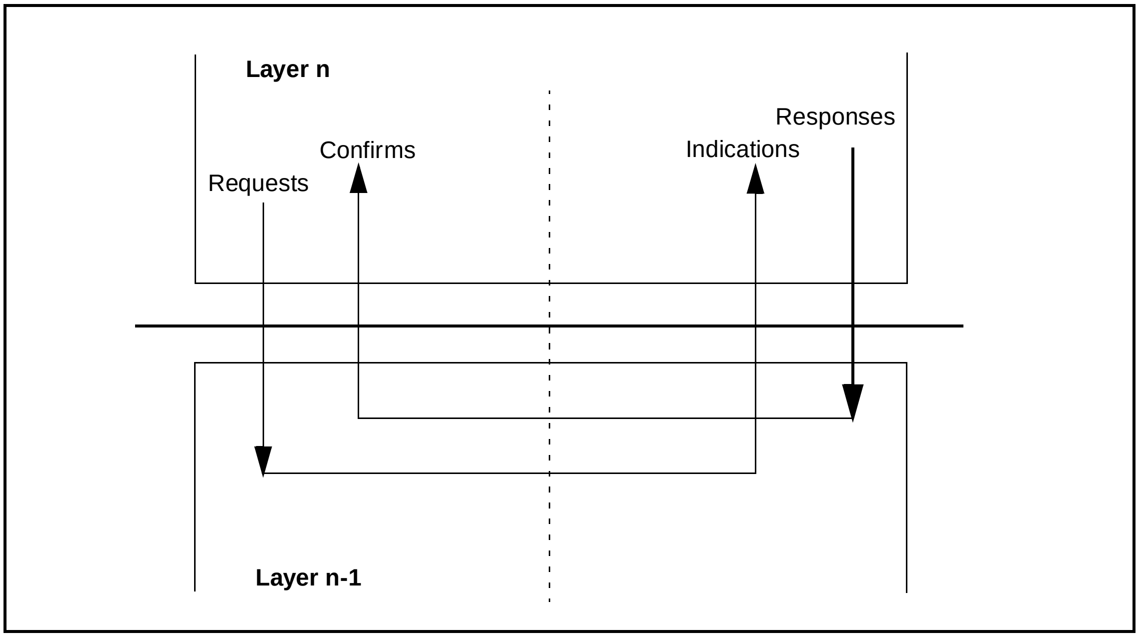


Figure 2-4: Primitives

- ISUP, PSIF - ISUP, and the layer manager are the service user of the PSF - ISUP layer. PSF - ISUP does not have any service provider.

- On receipt of run-time update request from the portable ISUP and/or PSIF - ISUP, the active PSF - ISUP sends a data request, and the standby PSF - ISUP receives data indications. The standby PSF - ISUP does not send any explicit response during run-time update.

- During warmstart and controlled switchover updates, the active PSF - ISUP sends a data request, and the standby PSF - ISUP receives data indications. After receiving the last update message from active, the standby PSF - ISUP sends a response that acts as a confirmation for active PSF - ISUP.

Software Organization

PSF - ISUP Stack Architecture

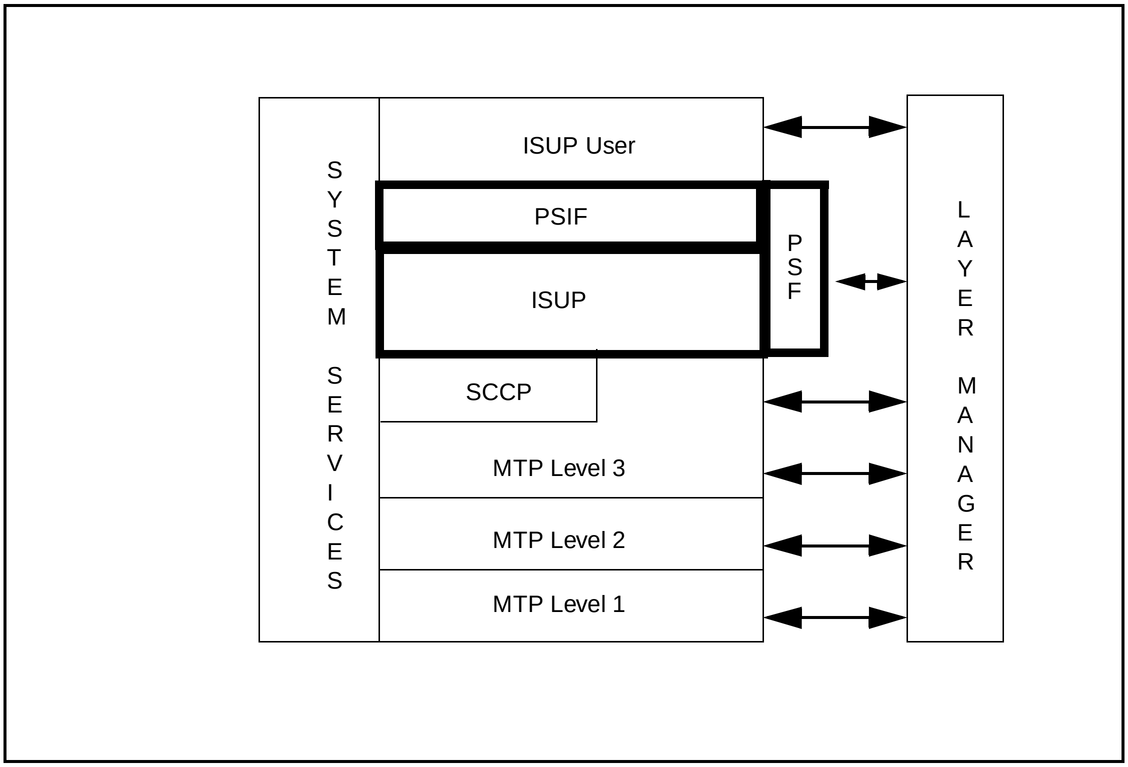


Figure 2-5: PSF - ISUP stack architecture

Software Organization

PSF - ISUP Stack Architecture

- ISUP user

  Application entity (for example, call control)

- ISUP

  Standard implementation of the following recommendations:

  - 1988 ITU Q.761-Q.764
  - 1992 ITU Q.761-Q.764, Q.767, Q.766, and Q.730
  - 1988 and 1992 ANSI T1.113
  - Singapore Telecom SS7 Specification
  - ETSI ETS 300-356
  - Italian Telecom ISUP-S
  - German Telekom FTZ 163-TR-75.95
  - Bellcore GR-317 and GR 394
  - NTT (Japan) Q761a , Q762a, Q763a , Q764a
  - NTT (Japan) Appendix 4 and Appendix 5

- PSIF - ISUP

  ISUP Wrapper that resides between Call Control and ISUP. PSIF - ISUP provides a generic interface between call control and ISUP.

- PSF - ISUP

  Implementation of FT/HA functions to Trillium's ISUP implementation

- SCCP

  Standard implementation of Signalling Connection Control Part Protocol. This layer provides services for sequenced connectionless and connection-oriented data transfer, using the services provided by the MTP layer.

Software Organization

PSF - ISUP Stack Architecture

- MTP Level 3

  Standard implementation of the Message Transfer Part - Level 3 of the SS7 stack. This layer provides routing, traffic management, link management, and route management functions.

- MTP Level 2 / MTP Level 1

  Standard implementation of the Message Transfer Part - Levels 1 and 2 of the SS7 stack. Level 2 provides reliable data transfer with the peer, while level 1 is the physical layer.

- Layer manager

  Provides the functions to control and monitor the condition of the protocol layer

- System services - operating system

  Provides the functions for buffer management, timer management, date and time management, resource checking, and initialization

Software Organization

Definitions

The table below defines terms used throughout this document:

Term	Definition
Active node	A node that executes software to provide the necessary protocol functionality. The active node processes the protocol messages and updates the new state information in the standby node.
Controlled switchover	A procedure that makes a standby node active and an active node standby
Fault-tolerant node	A pair of nodes with replicated protocol layers. A fault-tolerant node can have an active, standby, or OOS state.
Forced switchover	A procedure that makes a standby node active when an active node goes OOS
Node	A unit that has a processor(s) with private volatile memory inaccessible to all other nodes, and a private clock governing the execution of instructions on this processor. A node also has a network interface connecting it to a communication network using communication channels. The software governs the sequence of instructions executed on a node.
OOS node	An off-line node that has the ability to become either an active or standby node
Run-time state update	The active ISUP (PSIF - ISUP) handles protocol events that can result in internal ISUP (PSIF - ISUP) state changes. The active PSF updates the standby with the state changes to keep the standby synchronized.
Standby node	A node that acts as a backup to an active node
Warmstart	A procedure that makes an OOS node standby. An active node updates this new standby node with current information using a bulk update procedure

3 FILES

- Product files

- Common files

- Sample files

- Document files

- Organization
  - ISUP file organization
  - PSF - ISUP file organization

Files

Product Files

Name	Description
zi.h	PSF - ISUP - Defines
zi_err.h	PSF - ISUP - Error - Defines
zi.x	PSF - ISUP - **Typedef**s, variables, prototypes
zi_bdy1.c	PSF - ISUP - Body file 1 - Interface primitives
zi_bdy2.c	PSF - ISUP - Body file 2- Support functions
zi_bdy3.c	PSF - ISUP - Body file 3- Update message encoding functions
zi_bdy4.c	PSF - ISUP - Body file 4- Update message decoding functions
zi_id.c	PSF - ISUP - ID
zi_ex_ms.c	PSF - ISUP- External interface
zi_ptpi.c	PSF - ISUP - Portable peer interface
zi_ptmi.c	PSF - ISUP - Portable stack manager interface
zi.mak	PSF - ISUP - **Makefile**
zi_pt.lnk	PSF - ISUP - Portable - Link
m600.bat	Command file - Make portable software for DOS
c600.bat	Command file - Make codeview compatible portable software for DOS
msun	Command file - Make portable software for Solaris
1016148.asc	Release Notes - ASCII text
1093146.asc	Portation Guide - ASCII text

Files

Common Files

Name	Description
envopt.h	Environment option
envdep.h	Environment-dependent
envind.h	Environment-independent
gen.h	General - Defines
ssi.h	System services interface - Defines
cm_pftha.h	Common PSF - ISUP - Defines
lzi.h	PSF - ISUP layer manager interface - Defines
gen.x	General - **Typedef**s, prototypes
ssi.x	System services interface - **Typedef**s, prototypes
cm_pftha.x	Common PSF - ISUP lower - **Typedef**s, prototypes
lzi.x	PSF - ISUP layer manager interface - **Typedef**s, prototypes
cm5.h	Common - Timer functions - Defines
cm_ss7.h	Common - SS7 packing/unpacking functions - Defines
cm_hash.h	Common hash functions - Defines
cm5.x	Common - Timer functions - **Typedef**s, prototypes
cm_ss7.x	Common - SS7 packing/unpacking functions - **Typedef**s, prototypes
cm_hash.x	Common hash - **Typedef**s, prototypes
cm_lib.x	Common library - **Typedef**s, prototypes
cm_bdy5.c	Common - Timer functions
cm_gen.c	Common - General packing / unpacking functions
cm_ss7.c	Common - SS7 packing/unpacking functions
cm_hash.c	Common hash functions
cm_lib.c	Common library functions
cm_pftha.c	Common - PSF - ISUP packing/unpacking functions
ss_ptsp.c	System services - Portable service provider

Files

Sample Files

Name	**Description**
`zi_acc.h`	PSF - ISUP - Sample - Acceptance test - Defines
`zi_acc.x`	PSF - ISUP - Sample - Acceptance test - **Typedef**s and prototypes
`zi_acc1.c`	PSF - ISUP - Sample - Acceptance test - Main test driver
`zi_acc2.c`	PSF - ISUP - Sample - Acceptance test - Test cases
`smzi_err.h`	PSF - ISUP - Sample - PSF - ISUP stack manager - Error defines
`sm_bdy1.c`	PSF - ISUP - Sample - Global stack manager - Body - Part 1
`smzibdy1.c`	PSF - ISUP - Sample - PSF - ISUP stack manager - Body - Part 1
`smziptmi.c`	PSF - ISUP - Sample - PSF - ISUP stack manager - Portable management interface
`sm_ex_ms.c`	PSF - ISUP - Sample - Global stack manager - External interface
`smziexms.c`	PSF - ISUP - Sample - PSF - ISUP stack manager - External interface
`zi.mak`	PSF - ISUP - **Makefile**
`zi_acc.lnk`	PSF - ISUP - Sample - **Linkfile**
`msunacc`	Command file - Make acceptance software for Solaris

Note: *The above table lists PSF - ISUP sample files only. PSF - ISUP sample files should be used along with ISUP product sample files.*

Files

Document Files

Name	Description
1091133	*Fault-Tolerant/High-Availability (FT/HA) Core Functional Specification*
1092133	*Fault-Tolerant/High-Availability (FT/HA) Core Service Definition*
1091146	*PSF - ISUP Functional Specification*
1092146	*PSF - ISUP Service Definition*
1094146	*PSF - ISUP Software Test Sample*
1095146	*PSF - ISUP Training Course*
1111001	*System Services Interface Service Definition*

Files

Organization

PSF - ISUP file organization is coupled with ISUP (PSIF - ISUP) product file organization. Various stack components are organized into files according to the following conditions:

- The ISUP protocol layer is represented by **`si_bdy[?].c`** (see the *ISUP Portation Guide*)
- The ISUP user is represented by **`layer5.c`** (see the *ISUP Portation Guide*)
- The MTP Level 3 layer is represented by **`layer3.c`** (see the *ISUP Portation Guide*)
- The PSF - ISUP layer is represented by **`zi_bdy[?].c`**
- The layer manager is represented by **`smzibdy1.c`**
- The system services provider is represented by **`ss_ptsp.c`**
- PSIF - ISUP layer is represented by **`iw_bdy[?].c`** (see the *PSIF - ISUP Service Definition*)

Files

Organization

ISUP File Organization

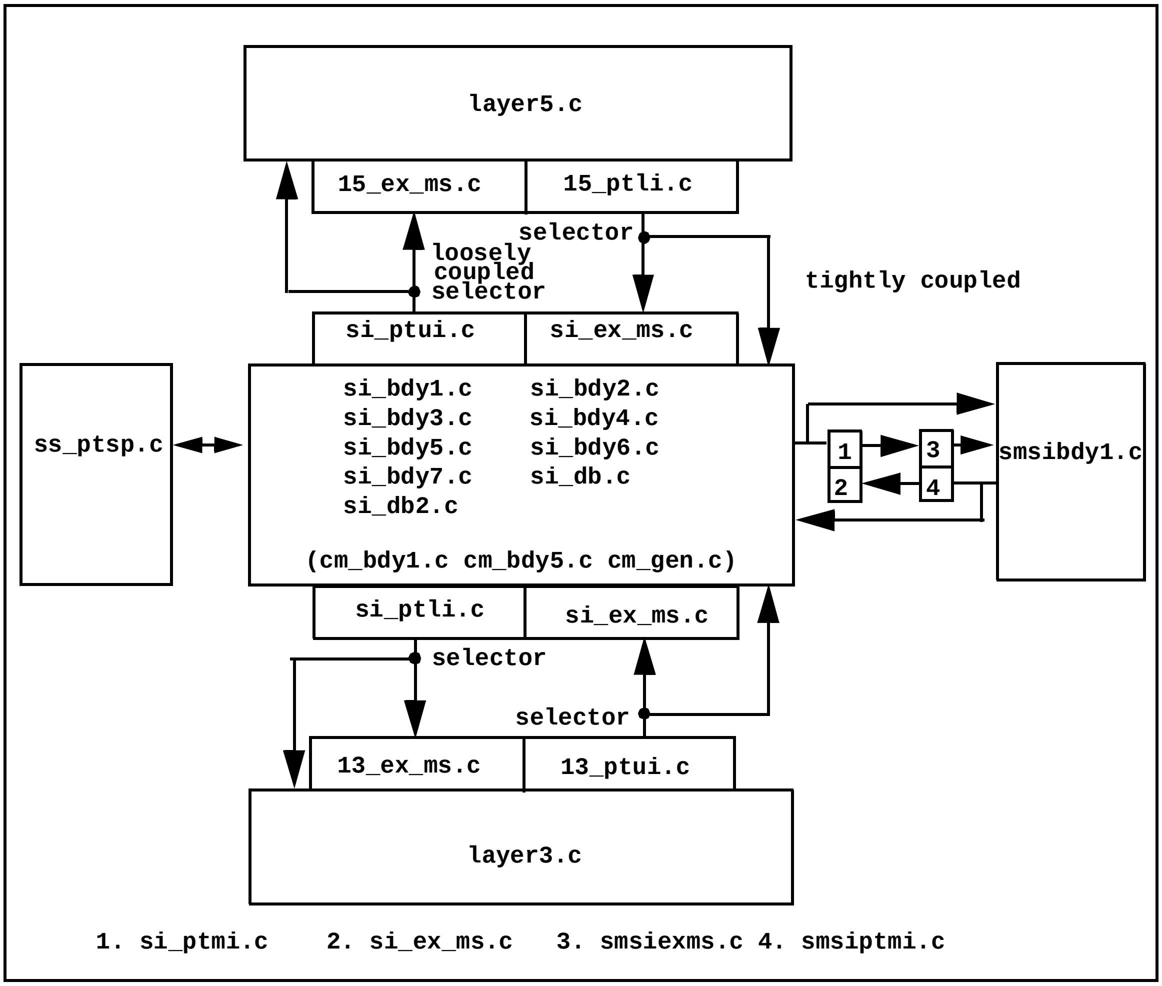


Figure 3-1: ISUP file organization

Files

Organization

PSF - ISUP File Organization

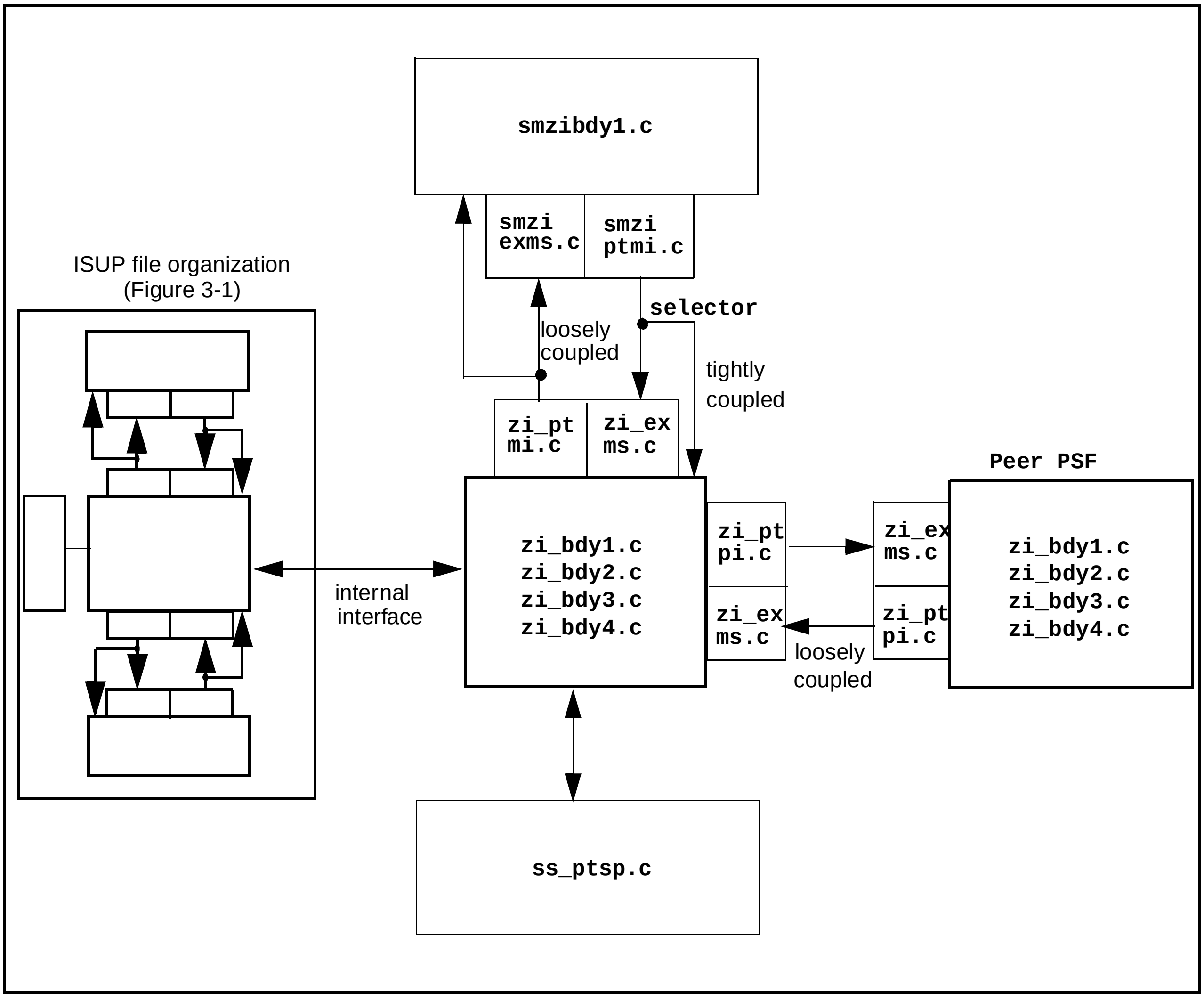


Figure 3-2: PSF - ISUP file organization

Note: `si` *management interface is not shown in the above figure.*

4 FUNCTIONAL DESCRIPTION

- Block diagram

- Service characteristics
  - General
  - Fault-Tolerant / High-Availability (FT / HA)
  - Layer manager interface

- Update messages

- Performance characteristics

- Maximum configuration

Functional Description

Block Diagram

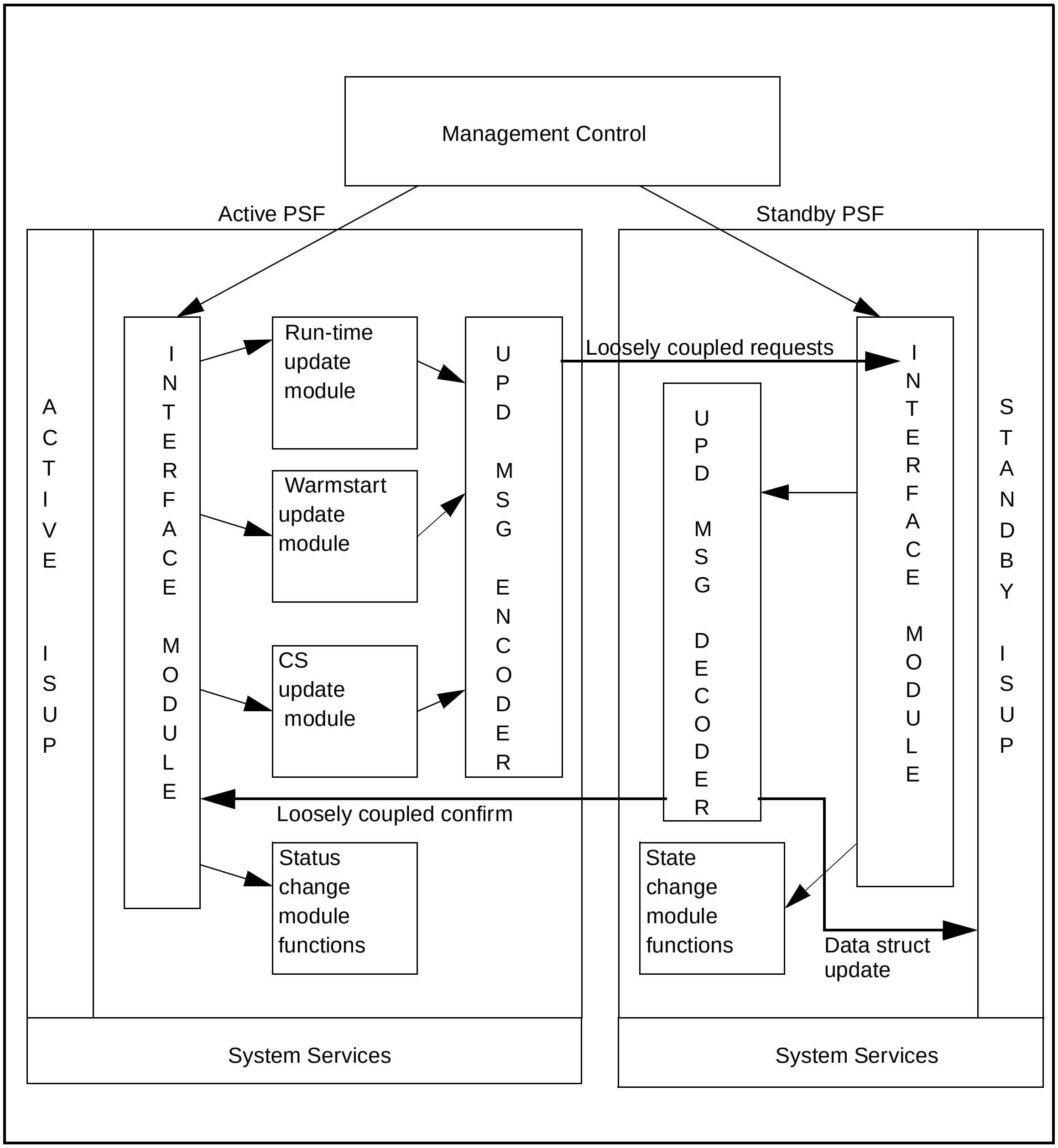


Figure 4-1: PSF - ISUP functional description

Functional Description

Service Characteristics

General

- Provides Fault-Tolerant / High-Availability (FT / HA) capability in Trillium's ISUP and PSIF - ISUP products in a modular fashion

- Hides most of the fault-tolerant details from the portable ISUP layer, thereby preserving the existing architecture of the ISUP product. It also provides fault-tolerance for the PSIF - ISUP software, transparent to the working of the PSIF layer.

- Uses a state update approach to keep the active and standby ISUP synchronized

- Handles various switchover commands from management to recover from faults

- Supports the following recommendations:
    - 1988 ITU Q.761-Q.764
    - 1992 ITU Q.761-Q.764, Q.767, Q.766, and Q.730
    - 1988 and 1992 ANSI T1.113
    - Singapore Telecom SS7 Specification
    - ETSI ETS 300-356
    - Italian Telecom ISUP-S
    - German Telekom FTZ 163-TR-75.95
    - Bellcore GR-317 and GR 394
    - NTT (Japan) Q761a , Q762a, Q763a , Q764a
    - NTT (Japan) Appendix 4 and Appendix 5

Functional Description

Service Characteristics

Fault-Tolerant/High-Availability

- Supports warmstart update of the peer ISUP (PSIF - ISUP) for making it standby from Out-Of-Service (OOS). Only stable states are updated during the warmstart update.

- Supports the ability to let ISUP (PSIF - ISUP) handle protocol events during the warmstart. Since the active PSF - ISUP performs the warmstart state update in multiple scheduling, the protocol events can be handled in-between by the active PSF - ISUP.

- Supports run-time state update to keep the standby and active PSF - ISUP synchronized. Only stable states are updated at run time.

- Supports no loss of states during controlled switchover. The transient states are updated to the standby before the switchover.

- Supports forced switchover when the active PSF - ISUP becomes OOS. The standby becomes active and PSF - ISUP resumes operation with its current stable states.

- Supports the functionality to abort an ongoing warmstart state update or controlled switchover state update.

- Supports the disabling of run-time state update from the active node to the standby node when the standby becomes OOS.

- Supports shutdown procedure, which resets the PSF - ISUP states and deallocates the memory allocated by the PSF - ISUP for its operation.

- Multiple Originating Point Code (OPC) implementation in ISUP functionality is also reflected on the standby with no loss of point code information, status, and accessibility.

- Detects the standby ISUP (PSIF - ISUP) going out of synchronization.

Functional Description

Service Characteristics

Layer Manager Interface

- Allows configuration and control from the layer manager
- Provides interface to the layer manager for the gathering of status information
- Generates alarms to the layer manager when an abnormal condition occurs, such as a memory allocation failure
- Supports debug printing, which can be controlled from the layer manager interface

Functional Description

Update Messages

The active PSF - ISUP sends update messages to the standby PSF - ISUP to keep the standby synchronized with the active at run time. The general characteristics of an update message are:

- All update messages are sequenced for error detection on the standby PSF - ISUP. The active PSF - ISUP uses a separate sequence number for the run-time update and bulk update (warmstart and controlled switchover update) message.
- Each update message has one of the following update types:
  - Run-time
  - Warmstart
  - Controlled switchover (synchronization)
- All bulk update messages have an indicator field that denotes if more messages of a similar type follow this update message.
- A single update message can carry information about multiple ISUP (PSIF - ISUP) control blocks. A single control block inside an update message is represented by a table type (control block type), followed by the actual control block information to be updated.

Functional Description

Performance Characteristics

- PSF - ISUP can be configured for a maximum update message size (subjected to a minimum value) that PSF - ISUP can generate for run-time updates. Other system parameters (such as buffers and communication drivers) can be adjusted for better performance at run time. (See **`maxUpdMsgSize`** in the **`cfg`** structure, which comes as an argument of the **`ZiMiLziCfgReq`** primitive from the layer manager)

- At run time, PSF - ISUP only updates the stable states to stand by in order to keep the update overhead to a minimum.

- PSF - ISUP provides better system availability by supporting warmstart update in parallel with the processing of protocol events.

- PSF - ISUP is designed not to starve other system processes during controlled switchover and warmstart update.

Functional Description

Maximum Configuration

- PSF - ISUP supports only 1 peer SAP. For information about the peer, see Section 6, "INTERNAL ORGANIZATION."

- The maximum size of an update message: 2^{32}

Note: *The actual maximums depend on available system memory.*

5 INTERFACES

- Trillium Advanced Portability Architecture (TAPA)
- General description
- System services interface
- Layer manager interface
- Peer layer interface
- Protocol layer interface
- Protocol Specific Interface Function interface

Interfaces

Trillium Advanced Portability Architecture

PSF - ISUP

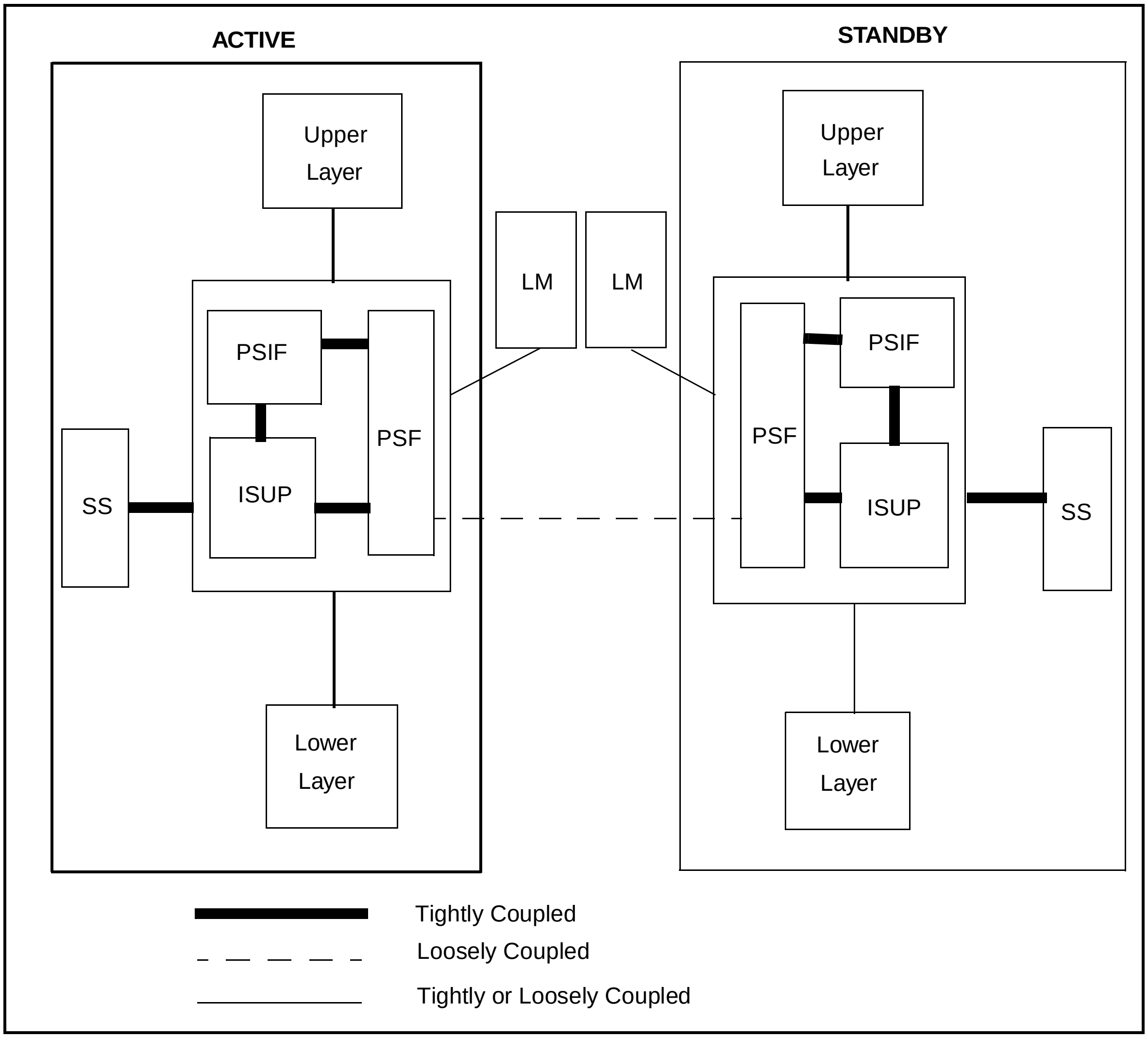


Figure 5-1: TAPA: PSF - ISUP

Interfaces

General Description

- System services

  Functions required by the protocol layer for buffer management, timer management, date and time management, resource checking, and initialization

- Layer manager

  Functions to control and monitor the condition of the protocol layer

- Peer layer interface

  Functions required by PSF - ISUP to interface with the peer PSF - ISUP for updating control block state change information

- Protocol layer interface

  Interface between the ISUP protocol layer and PSF - ISUP

- Protocol Specific Interface Function interface

  Interface between the PSIF - ISUP protocol layer and PSF - ISUP

Interfaces

System Services - General

- Task manager

  Functions to initialize and schedule protocol layer

- Timer manager

  Functions to provide periodic activation of protocol layer timer functions

- Memory manager

  Functions to allocate and deallocates memory and to create, add, and remove data from messages

- Queue manager

  Functions to create, add, and remove messages or data from queues

Interfaces

System Services - Specific

PSF - ISUP Specific

- PSF - ISUP is not registered as a separate entity with system services. PSF - ISUP assumes the same entity and instance ID as the associated ISUP protocol layer and PSIF - ISUP layer.

- Other entities (such as the layer manager, peer PSF - ISUP, or PSIF - ISUP) use the ISUP protocol layer entity and instance IDs to communicate, via a loosely coupled interface, with PSF - ISUP.

- The PSF - ISUP task initialization function and activation functions are not registered with system services. These functions are internally invoked via the ISUP protocol layer's initialization and activation functions.

- PSF - ISUP registers a timer activation function with system services. System services is responsible for periodically calling the timer activation function.

Interfaces

System Services - Specific

Initialization

- **`ziActvInit`**

  Initialization function for PSF - ISUP. This function is provided in PSF - ISUP and must be invoked before PSF - ISUP starts operating. This function initializes internal structures and is invoked internally by the ISUP protocol layer initialization function (**`siActvInit`**).

```
PUBLIC S16 ziActvInit(ent, inst, region, reason)
Ent ent;                /* entity */
Inst inst;              /* instance */
Region region;          /* region */
Reason reason;          /* reason */
```

- The data flow is:

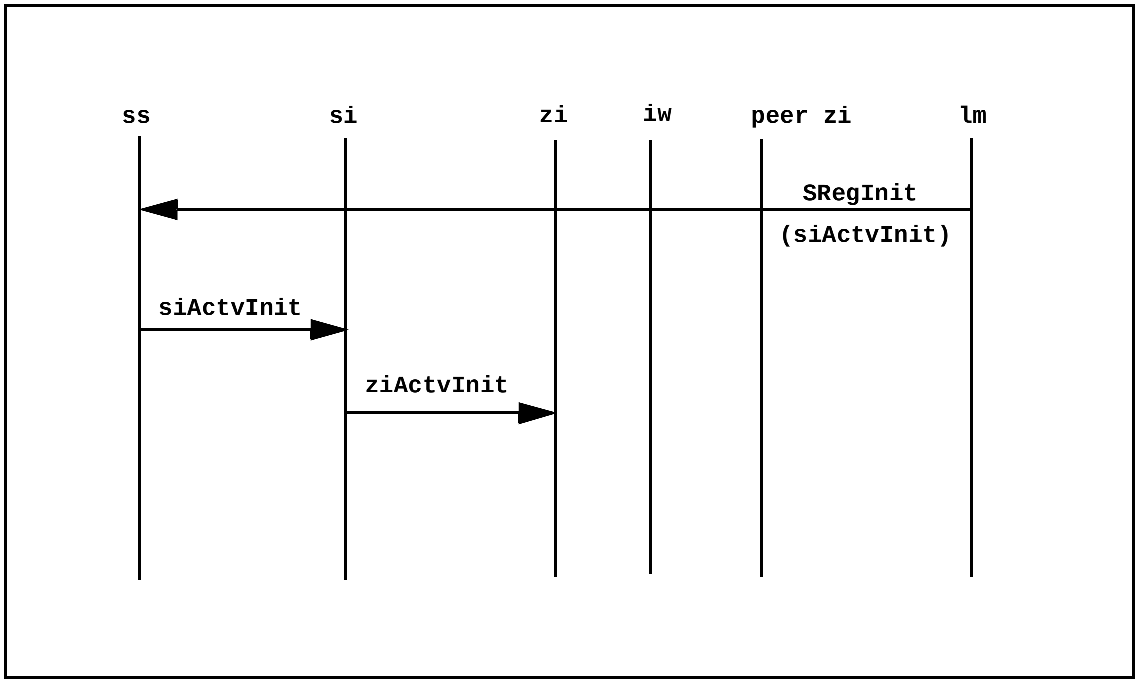


Figure 5-2: Data flow—system services–initialization

Interfaces

System Services - Specific

Task Management

- **ziActvTsk**

  Layer activation function for PSF - ISUP. This function is only used for loosely coupled interfaces and is not registered with system services. This function is internally invoked by the ISUP protocol layer activation function (**siActvTsk**).

```
PUBLIC S16 ziActvTsk(pst, mBuf)
Pst      *pst;            /* post */
Buffer   *mBuf;           /* packed primitive */
```

- The data flow is:

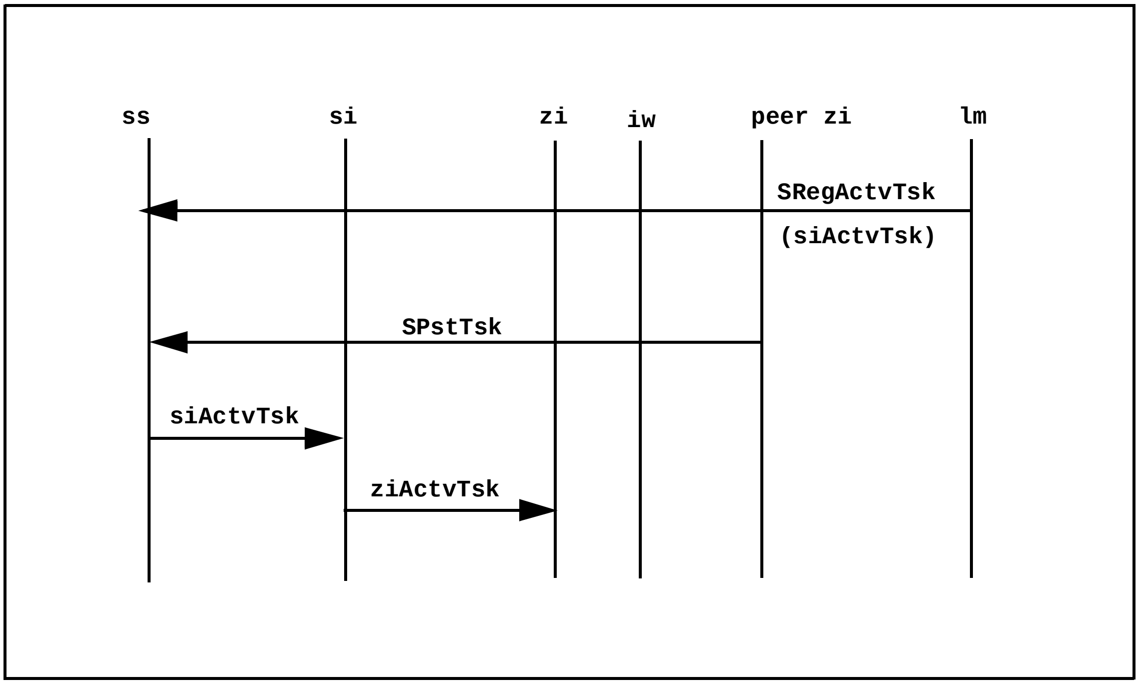


Figure 5-3: Data flow—system services–task management

Interfaces

System Services - Specific

Timer Management

- **`ziActvTmr`**

  Called by system services to provide a timer tick so that PSF - ISUP can maintain its own timers. The period between activations is registered using **`SRegTmr.`**

  `PUBLIC S16 ziActvTmr()`

- The data flow is:

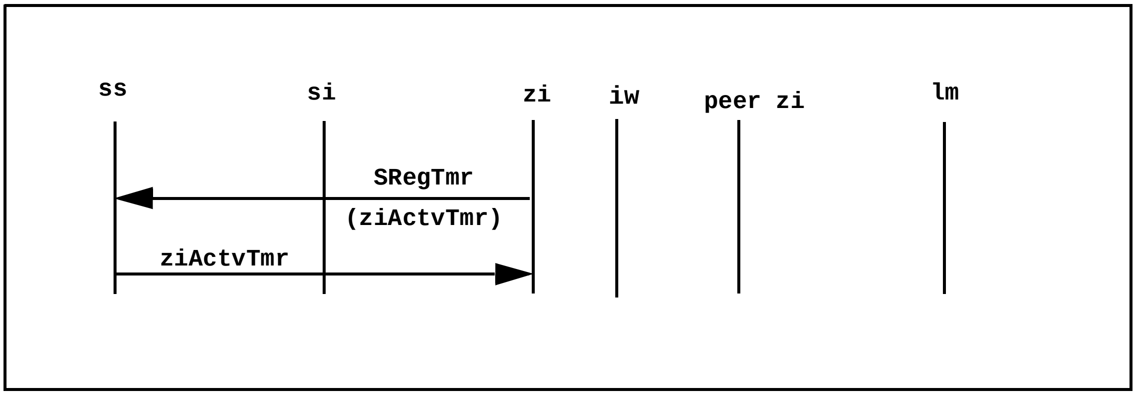


Figure 5-4: Data flow—system services–timer management

Interfaces

System Services - Specific

Post Structure

```
typedef struct pst                 /* parameters for SPstTsk */
{
   ProcId    dstProcId;            /* destination processor id */
   ProcId    srcProcId;            /* source processor id */
   Ent       dstEnt;               /* destination entity */
   Inst      dstInst;              /* destination instance */
   Ent       srcEnt;               /* source entity */
   Inst      srcInst;              /* source instance */
   Prior     prior;                /* priority */
   Route     route;                /* route */
   Event     event;                /* event */
   Region    region;               /* region */
   Pool      pool;                 /* pool */
   Selector  selector;             /* selector */
   U16       spare1;               /* spare 1 */
} Pst;
```

- **pst**

  Post structure. Used to route the primitive from the calling layer to the called layer.

  All primitives have the post structure as their first parameter. **Pst** routes the primitive from the source layer to the destination layer. Normally, the interface between layers can either be loosely or tightly coupled.

  pst->selector resolves the primitive at the calling layer. It determines the memory region and pool from which any needed message buffer is to be allocated. In addition, it determines the priority and route for the message and specifies the source and destination entities.

  Once the primitive reaches the destination layer, this parameter is no longer useful.

Interfaces

Layer Manager - General

- Configuration

  Configures PSF - ISUP's operational parameters

- Control

  Controls PSF - ISUP's activities

- Solicited status

  Indicates the current state of the PSF - ISUP

- Unsolicited status (alarm)

  Indicates a change in status of the PSF - ISUP

Interfaces

Layer Manager - General

File	Description
`ZiMiLziCfgReq`	Configure request
`ZiMiLziCfgCfm`	Configure confirm
`ZiMiLziCntrlReq`	Control request
`ZiMiLziCntrlCfm`	Control confirm
`ZiMiLziStaReq`	Status request
`ZiMiLziStaCfm`	Status confirmation
`ZiMiLziStaInd`	Unsolicited status indication (alarm)

Interfaces

Layer Manager - Concepts

- Each management primitive is called with two parameters. One is the post structure for routing the primitive. The second is of type **CmPFthaMngmt**. For example,

```
ZiMiLziCfgReq(pst, cfg)
Pst          *pst;      /* post structure */
CmPFthaMngmt *cfg;      /* configuration structure */

ZiMiLziStaReq(pst, sta)
Pst          *pst;      /* post structure */
CmPFthaMngmt *sta;      /* status structure */
```

- The **CmPFthaMngmt** structure has a header that identifies the target, followed by a union of all the desired management structures.

```
typedef struct CmPFthaMngmt  /* management structure */
{
   Header   hdr;             /* header */
   CmStatus cfm;             /* confirm */
   union
   {
      ...                    /* configuration */
      ...                    /* solicited status */
      ...                    /* unsolicited status */
      ...                    /* control */
   } t;
} CmPFthaMngmt;
```

Interfaces

Layer Manager - Concepts

- Header

```
typedef struct tds_header      /* header */
{
   U16       msgLen;           /* message length - optional */
   U8        msgType;          /* message type - optional */
   U8        version;          /* version - optional */
   U16       seqNmb;           /* sequence number - optional */
   EntityId  entId;            /* entity id - optional */
   ElmntId   elmId;            /* element id - mandatory */
#ifdef LMINT3
   TranId    transId;          /* transaction Id - mandatory */
   Resp      response;         /* response parameters - mandatory */
#endif /* LMINT3 */
} Header;
```

elmId

Identifies the element type to be managed.

```
STGEN                          /* general */
STPEERSAP                      /* peer SAP */
STSID                          /* system Id */
```

transId

Identifies the transaction ID used by the layer manager to sequence requests. PSF - ISUP sends the same transaction ID back to the layer manager in the confirm associated with the request.

response

Layer manager passes this structure in requests. PSF - ISUP fills the post structure for sending a confirm from the response values supplied in the management request. **response** has the following format:

```
typedef struct resp
{
   Selector   selector;        /* selector */
   Priority   prior;           /* priority */
   Route      route;           /* route */
   MemoryId   mem;             /* memory */
}Resp;
```

Interfaces

Layer Manager - Concepts

`selector`

Selector value to be used by PSF - ISUP to reply to the layer manager.

`0` - Loosely coupled

`1` - Tightly coupled

`prior`

Priority to be used by the PSF - ISUP inside the **`Pst`** structure while responding to a layer manager request.

`route`

Route to be used by the PSF - ISUP inside the **`Pst`** structure while responding to a layer manager request.

`mem`

Region and pool of the memory to be used by PSF - ISUP to send a confirmation back to the layer manager.

Interfaces

Layer Manager - Concepts

- **CmStatus**

  PSF - ISUP uses the status structure in confirms to indicate the result of a request and to identify the reason for any failures. This field is used only in the confirm primitives going from PSF - ISUP to the stack manager.

```
typedef struct cmStatus
{
   U16 status;                /* status of request */
   U16 reason;                /* failure reason */
}CmStatus;
```

status

Status to indicate the success or failure of a layer manager request coming to PSF - ISUP.

```
LCM_PRIM_OK                   /* success*/
LCM_PRIM_NOK                  /* failure */
LCM_PRIM_OK_NDONE             /* activity in progress */
```

reason

Reason for the failure when a layer manager request is not successful (that is, when the status value is **LCM_PRIM_NOK**). The value for this field depends on the request primitive type.

Interfaces

Layer Manager - Specific

Configuration Request

```
PUBLIC S16 ZiMiLziCfgReq(pst, cfg)
Pst           *pst;               /* post structure */
CmPFthaMngmt  *cfg;               /* configuration */
```

- The object to be configured is identified in the header (**hdr.elmId.elmnt**).

```
STGEN                             /* general */
STPEERSAP                         /* peer SAP */
```

- **cfg**

```
typedef struct CmPFthaMngmt
{
   Header    hdr;                 /* header */
   CmStatus  cfm;                 /* confirm */
   union
   {
   .
   .
   .
      struct
      {
         union
         {
            CmPFthaGenCfg genCfg;     /* general configuration */
            CmPFthaSAPCfg peerSAPCfg; /* peer SAP configuration */
         } s;
      } cfg;                          /* configuration */
   .                                  /* solicited status */
   .                                  /* unsolicited status */
   .                                  /* control */
   } t;
} CmPFthaMngmt;
```

Interfaces

Layer Manager - Specific

Configuration Request - General

hdr.elmId.elmnt is **STGEN**

Configures the timer resolution in ticks. This is the period between successive activations of protocol timer functions. Thus, this value would typically be the greatest common denominator of all protocol timer values in ticks.

Configures the post structure (for example, the processor ID) so that PSF - ISUP can send unsolicited status indications (alarms) to the layer manager.

Configures the virtual processor ID of the node

Configures the memory region and pool to allocate memory for posting a message to itself

```
typedef struct cmPFthaGenCfg
{
   S16      timeRes;     /* timer resolution */
   ProcId   vProcId;     /* virtual processor Id */
   MemoryId mem;         /* self region and pool */ /* reconfigurable */
   Pst      smPst;       /* stack manager post structure */ /* reconfig */
}CmPFthaGenCfg;
```

Note: *General configuration of PSF - ISUP must be done after the general configuration of the ISUP protocol layer and the PSIF - ISUP layer.*

Interfaces

Layer Manager - Specific

Configuration Request - Peer SAP

`hdr.elmId.elmnt` is **`STPEERSAP`**

Configures the post structure information for communicating with the peer PSF - ISUP. Since the interface with the peer PSF - ISUP is always loosely coupled, the **`selector`** field is unused.

Configures the maximum update message size that PSF - ISUP can build for a run-time update. The minimum allowable value is compiler-dependent. The recommended minimum value is 200. The maximum value depends on underlying system services. PSF - ISUP update messages can exceed the configured size during controlled switchover update.

Configures the update acknowledgment timer. The recommended value is 5 to 10 seconds. This value must be specified in terms of the time resolution provided during general configuration.

For example, if the value of timer **`tUpdCompAck`** needs to be 5 seconds, the system tick is 1/10th of a second, the configured timer resolution is 10, and the configured value is 5.

Desired Timer Value = System Tick × Timer Resolution × Configured Value

5 (seconds) = 0.1 × 10 × 5

Interfaces

Layer Manager - Specific

Configuration Request - Peer SAP

All parameters within the structure given below are reconfigurable.

```
typedef struct cmPFthaSAPCfg
{
   Region     region;          /* memory region for peer */
   Pool       pool;            /* memory pool for peer */
   ProcId     dstProcId;       /* peer processor id */
   Ent        dstEnt;          /* peer entity id */
   Inst       dstInst;         /* peer instance id */
   Priority   prior;           /* peer post priority */
   Route      route;           /* peer post route */
   Selector   selector;        /* peer selector */
   TmrCfg     tUpdCompAck;     /* update completion timer */
   U32        maxUpdMsgSize;   /* maximum size of the update message */
} CmPFthaSAPCfg;


    typedef struct tmrCfg      /* timer configuration */
    {
       Bool   enb;             /* enable */
       U16    val;             /* value */
    } TmrCfg;
```

Note: *General configuration of PSF - ISUP must be done before peer SAP configuration.*

Interfaces

Layer Manager - Specific

Configuration Confirm

```
PUBLIC S16 ZiMiLziCfgCfm(pst, cfm)
Pst          *pst;         /* post structure */
CmPFthaMngmt *cfm;         /* confirm */
```

- Configuration confirm is sent to the originator of the configuration request. PSF - ISUP uses the originator's entity ID, instance ID, **procId**, and **hdr.response** in the configuration request to prepare the post structure for sending a configuration confirm.

- **cfm**

```
typedef struct CmPFthaMngmt
{
   Header   hdr;       /* header */
   CmStatus cfm;       /* confirm */
   union               /* not used for confirm*/
   {
   .
   .
   .
   } t;
} CmPFthaMngmt;
```

hdr.transId is the same as that received in the configuration request from the layer manager.

```
typedef struct cmStatus
{
   U16 status;     /* status of request */
   U16 reason;     /* failure reason */
}CmStatus;

status

LCM_PRIM_OK        /* success*/
LCM_PRIM_NOK       /* failure */
```

Interfaces

Layer Manager - Specific

Configuration Confirm

reason

```
LCM_REASON_NOT_APPL           /* Not applicable */
LCM_REASON_GENCFG_NOT_DONE    /* General configuration not done */
LCM_REASON_REGTMR_FAIL        /* Timer registration failed */
LCM_REASON_INV_PAR_VAL        /* Upd msg size parameter invalid */
LCM_REASON_QINIT_FAIL         /* Queue init failed */
LCM_REASON_INVALID_ELMNT      /* Invalid element in config req */
LCM_REASON_PRTLYRCFG_NOT_DONE /* Portable layer configuration not
                                 done */
```

Interfaces

Layer Manager - Specific

Configuration - Procedure

- The data flow is:

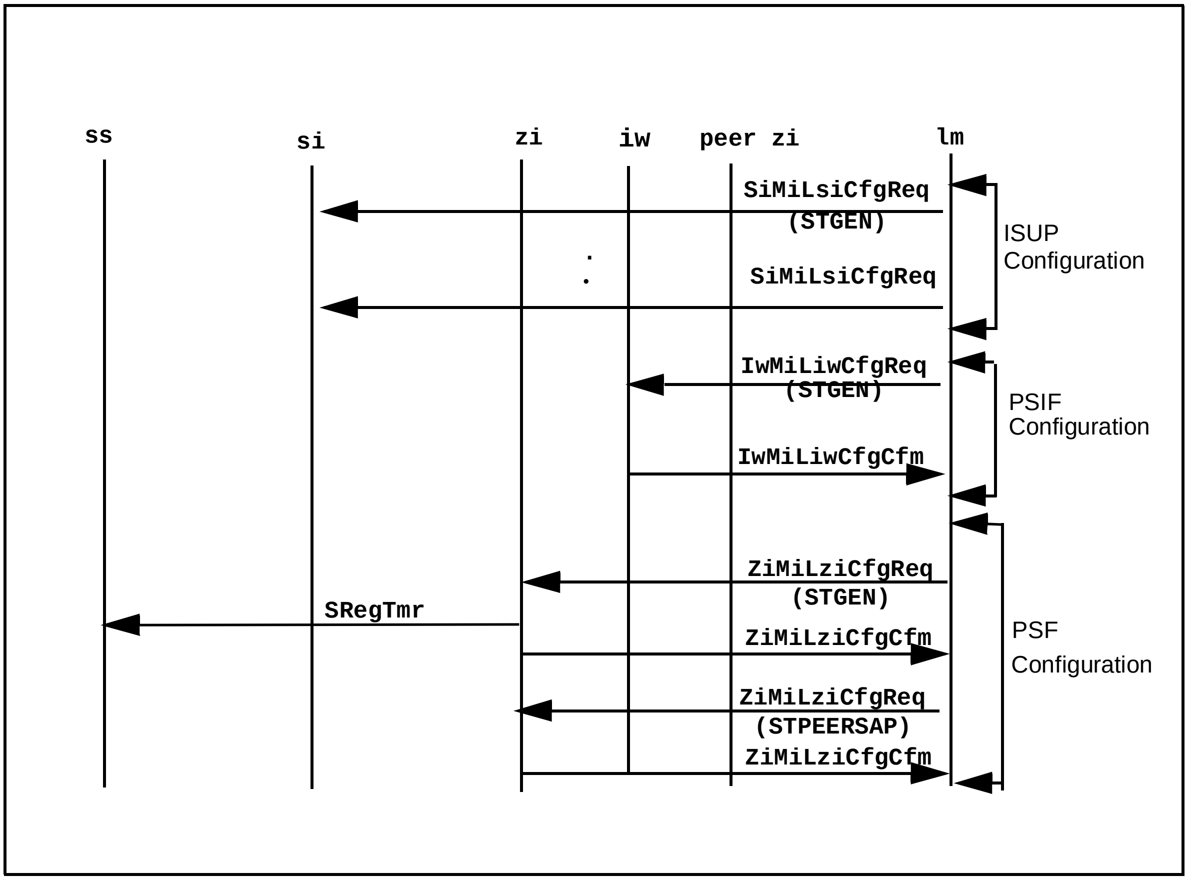


Figure 5-5: Data flow—configuration procedure

Interfaces

Layer Manager - Specific

Status Request

```
PUBLIC S16 ZiMiLziStaReq(pst, sta)
Pst          *pst;       /* post structure */
CmPFthaMngmt *sta;       /* status structure */
```

- The header (**hdr.elmId.elmnt**) identifies the object whose status is being requested.
    - **STGEN**
    - **STPEERSAP**
    - **STSID**

- **sta.t.sta** structure is returned with values in status confirm.

- **sta**

```
typedef struct cmPFthaMngmt
{
   Header   hdr;      /* header */
   CmStatus cfm;      /* confirm */
   union              /* values are returned in sta structure*/
   {
    .
    .
    .
   } t;
 } CmPFthaMngmt;
```

Interfaces

Layer Manager - Specific

Status Request - General

- **hdr.elmId.elmnt** is **STGEN**

- **genSta** returns the status of the ISUP protocol layer. Allowable values are:

```
ACTIVE              /* protocol layer active */
STANDBY             /* protocol layer standby */
OOS                 /* protocol layer out of service */
```

Interfaces

Layer Manager - Specific

Status Confirm

```
PUBLIC S16 ZiMiLziStaCfm (pst, sta)
Pst           *pst;
CmPFthaMngmt  *sta;
```

- Used to return status values (depending on the **`hdr.elmId.elmnt`** in the status request)

- **sta**

```
typedef struct cmPFthaMngmt
{
   Header   hdr;                              /* header */
   CmStatus cfm;                              /* confirm */
   union
   {
   .                                          /* configuration */
   .
   .
      struct
      {
         DateTime dt;
         union
         {
            U8 genSta;                        /* general status */
            CmPFthaPeerSapSta peerSapSta;     /* peer sap status */
            SystemId sysId;                   /* system id */
         }s;
     }sta;                                    /* solicited status */

   .                                          /* unsolicited status */
   .                                          /* control */
   .
   } t;
} CmPFthaMngmt;
```

`hdr.transId` is the same as that received in the status request from the layer manager.

Interfaces

Layer Manager - Specific

Status Confirm

```
typedef struct cmStatus
{
   U16 status;           /* status of request */
   U16 reason;           /* failure reason */
}CmStatus;

    status

    LCM_PRIM_OK          /* success*/
    LCM_PRIM_NOK         /* failure */

    reason

    LCM_REASON_NOT_APPL        /* Reason not applicable. Used with
                                  LCM_PRIM_PK */
    LCM_REASON_INVALID_ELMNT   /* invalid element in status req */
```

Interfaces

Layer Manager - Specific

Status Confirm - Peer SAP

hdr.elmId.elmnt is **STPEERSAP**

peerSapSta

```
typedef struct cmPFthaPeerSapSta
{
   U8 bndState;            /* bind status */
   U8 updState;            /* update status */
}CmPFthaPeerSapSta;
```

bndState

Peer SAP bind status. Allowable values are:

```
CMPFTHA_BND             /* bound */
CMPFTHA_UBND            /* unbound */
```

updState

Peer SAP update status. Allowable values are:

```
CMPFTHA_IDLE            /* no update going on */
CMPFTHA_WRMSTRT         /* warm start update in progress */
CMPFTHA_SYNC            /* sync update in progress */
CMPFTHA_WRMSTRT_WAIT    /* waiting for warm start ack from peer */
CMPFTHA_SYNC_WAIT       /* waiting for sync ack from peer */
```

Interfaces

Layer Manager - Specific

Status Confirm - System ID

hdr.elmId.elmnt is **STSID**

```
typedef struct systemId         /* system id */
{
   S16 mVer;                    /* main version */
   S16 mRev;                    /* main revision */
   S16 bVer;                    /* branch version */
   S16 bRev;                    /* branch revision */
   S8 *ptNmb;                   /* part number */
} SystemId;
```

Interfaces

Layer Manager - Specific

Solicited Status - Procedure

- The data flow is:

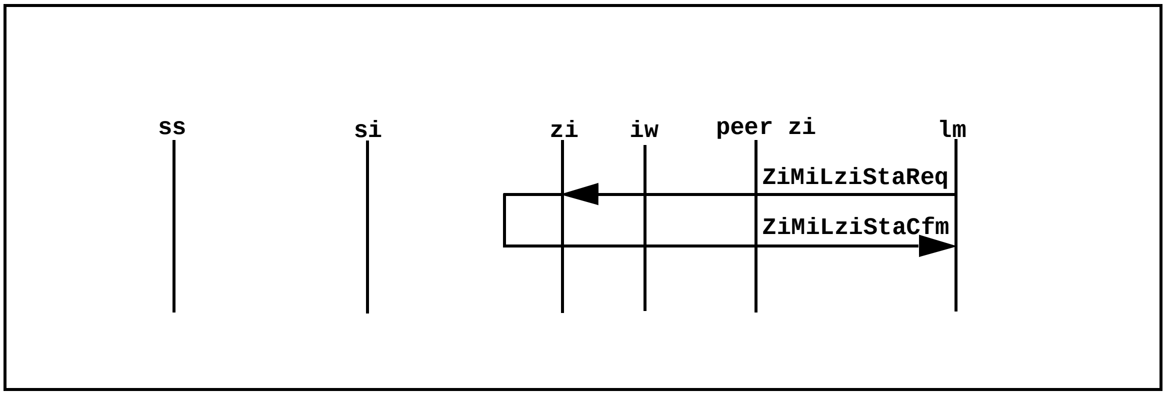


Figure 5-6: Data flow—solicited status procedure

Interfaces

Layer Manager - Specific

Unsolicited Status Indication

```
PUBLIC S16 ZiMiLziStaInd(pst, usta)
Pst          *pst;
CmPFthaMngmt *usta;
```

- Used to report error conditions and other major state changes to the layer manager

- **usta**

```
typedef struct cmPFthaMngmt
{
   Header   hdr;                              /* header */
   CmStatus cfm;                              /* confirm */
   union
   {
   .                                          /* configuration */
   .                                          /* solicited status */
   .
      struct
      {
         CmAlarm alarm;                       /* alarm structure */
         U8  evntParm[CMPFTHA_USTA_EP_MAX];   /* event parameters */
      }usta;                                  /* unsolicited status */
   .                                          /* control */
   .
   .
   } t;
} CmPFthaMngmt;
```

```
#define CMPFTHA_USTA_EP_MAX   16 /* length of eventparameter array */
```

alarm

```
typedef struct cmAlarm
{
   DateTime dt;        /* data and time */
   U16 category;       /* alarm category*/
   U16 event;          /* alarm event */
   U16 cause;          /* alarm cause */
}CmAlarm;
```

Interfaces

Layer Manager - Specific

Unsolicited Status Indication

Unsolicited status indication types:

category

```
LCM_CATEGORY_RESOURCE           /* resource category */
LCM_CATEGORY_PSF_FTHA           /* PSF category */
LCM_CATEGORY_INTERFACE          /* interface category */
```

event

```
CMPFTHA_MEM_FAILURE             /* memory allocation failure */
CMPFTHA_SEQERR                  /* sequence error on standby PSF */
CMPFTHA_UPDMSG_ERR              /* error in update message */
LCM_EVT_INV_TMR_EVT             /* invalid timer expiry */
```

cause

```
LCM_CAUSE_UNKNOWN               /* unknown cause */
LCM_CAUSE_PROT_NOT_ACTIVE       /* protocol layer not active */
```

Possible combinations of **category**, **cause**, and **event** of each alarm are shown in the table below:

Category	Event	Cause
LCM_CATEGORY_RESOURCE	CMPFTHA_MEM_FAILURE	CAUSE_UNKNOWN
LCM_CATEGORY_PSF_FTHA	CMPFTHA_SEQERR	CAUSE_UNKNOWN
LCM_CATEGORY_PSF_FTHA	CMPFTHA_UPDMSG_ERR	CAUSE_UNKNOWN
LCM_CATEGORY_INTERFACE	LCM_EVT_INV_TMR_EVT	LCM_CAUSE_PROT_NOT_ACTIVE

evntParm

Event parameters. This field provides more details about the alarm. Not used.

Interfaces

Layer Manager - Specific

Unsolicited Status Indication - Procedure

- The data flow is:

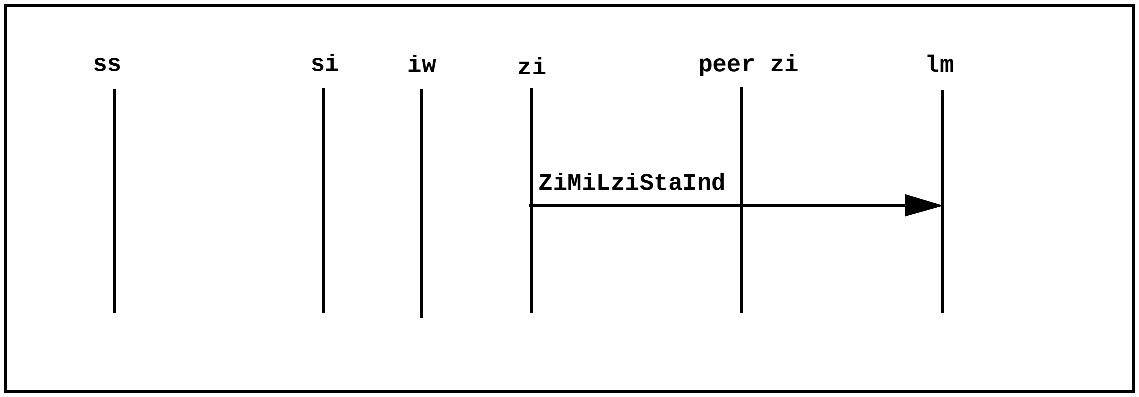


Figure 5-7: Data flow—unsolicited status indication procedure

Interfaces

Layer Manager - Specific

Control Request

```
PUBLIC S16 ZiMiLziCntrlReq(pst, cntrl)
Pst          *pst;
CmPFthaMngmt *cntrl;
```

- Used by the layer manager to control PSF - ISUP operation (for example, to make a node go from standby to active)

- **cntrl**

```
typedef struct cmPFthaMngmt
{
   Header   hdr;                        /* header */
   CmStatus cfm;                        /* confirm */
   union
   {
   .                                    /* configuration */
   .                                    /* solicited status */
   .                                    /* unsolicited status */
      struct
      {
         DateTime dt;                   /* date & time */
         U8 action;                     /* action */
         U8 subAction;                  /* subaction */
         union
         {
            CmPFthaDbgCntrl umDbg;      /* debug */
        }ctlType;
      }cntrl;                           /* control */
   .
   .
   .
   } t;
} CmPFthaMngmt;
```

```
typedef struct cmPFthaDbgCntrl
{
   U32 dbgMask;                         /* debug mask */
}CmPFthaDbgCntrl;
```

Interfaces

Layer Manager - Specific

Control Request

- Allowable values of various control request parameters are specified in the following table:

hdr.elmnt	Action	SubAction	Others	Purpose
STGEN	AENA	SAUSTA		Enable unsolicited status indications
		SADBG	dbgMask	Enable specified type of debug printing
	ADISIMM	SAUSTA		Disable unsolicited status indications
		SADBG	dbgMask	Disable specified type of debug printing
	AGO_ACT	SAENA_PEER_SAP or SADIS_PEER_SAP		Make PSF - ISUP active and enable / disable peer SAP
	AGO_SBY	SAENA_PEER_SAP		Make PSF - ISUP standby and enable peer SAP
	AWARMSTART	SAELMNT		Perform a warmstart update on standby ISUP (PSIF - ISUP)
	ASYNCHRONIZE	SAELMNT		Perform synchronization update on standby ISUP (PSIF - ISUP)
	AABORT	SAELMNT		Abort ongoing warmstart on synchronization update
	ASHUTDOWN			Completely shutdown PSF - ISUP operations and release all the resources
STPEERSAP	AUBND_DIS	SAELMNT		Unbind and disable peer SAP

Interfaces

Layer Manager - Specific

Control Request

- **dbgMask**

  dbgMask type values are:

  ```
  0                                /* all debug prints are disabled */
  DBGMASK_MI                       /* layer management interface */
  DBGMASK_PI                       /* peer interface */
  DBGMASK_PLI                      /* PSF protocol layer interface */
  LZT_DBGMASK_PACK                 /* pack mask */
  LZT_DBGMASK_UNPACK               /* unpack mask */
  LZI_DBGMASK_WARN                 /* Misc warning */
  ```

  dbgMask value can be a combination of defines values. For example, a valid value might be **(DBGMASK_MI | DBGMASK_PI)**.

- Control request retries are allowed. To avoid glare conditions when retrying a control request, it is recommended that a different **hdr.transId** value be used.

Interfaces

Layer Manager - Specific

Control Confirm

```
PUBLIC S16 ZiMiLziCntrlCfm(pst, cfm)
Pst           *pst;     /* post structure */
CmPFthaMngmt  *cfm;     /* confirm */
```

- Control confirm is sent to originator of the control request. PSF - ISUP uses originator's entity ID, instance ID, **procId**, and **hdr.response** in the control request to prepare the post structure for sending control confirm.

- **cfm**

```
typedef struct CmPFthaMngmt
{
   Header    hdr;     /* header */
   CmStatus  cfm;     /* confirm */
   union              /* not used for confirm*/
   {
   .
   .
   .
   } t;
 } CmPFthaMngmt;
```

hdr.transId is the same as that received in control request from the layer manager.

```
typedef struct cmStatus
{
   U16 status;       /* status of request */
   U16 reason;       /* failure reason */
}CmStatus;
```

Interfaces

Layer Manager - Specific

Control Confirm

status

```
#define LCM_PRIM_OK       0    /* success*/
#define LCM_PRIM_NOK      1    /* failure */
#define LCM_PRIM_OK_NDONE 2    /* accepted but not complete */
```

LCM_PRIM_OK_NDONE is followed by **LCM_PRIM_OK** for **AWARMSTART** and **ASYNCHRONIZE** actions.

reason

```
LCM_REASON_NOT_APPL            /* Reason not applicable: used with
                                  LCM_PRIM_PK */
LCM_REASON_INVALID_SUBACTION   /* invalid subaction */
LCM_REASON_INVALID_ACTION      /* invalid action */
LCM_REASON_PEER_SAP_NOT_CFG    /* peer sap not configured */
LCM_REASON_INVALID_STATE       /* control request in invalid
                                  state*/
LCM_REASON_INVALID_ELMNT       /* invalid header elmnt */
```

Interfaces

Layer Manager - Specific

Management Control - Procedure

- The data flow is:

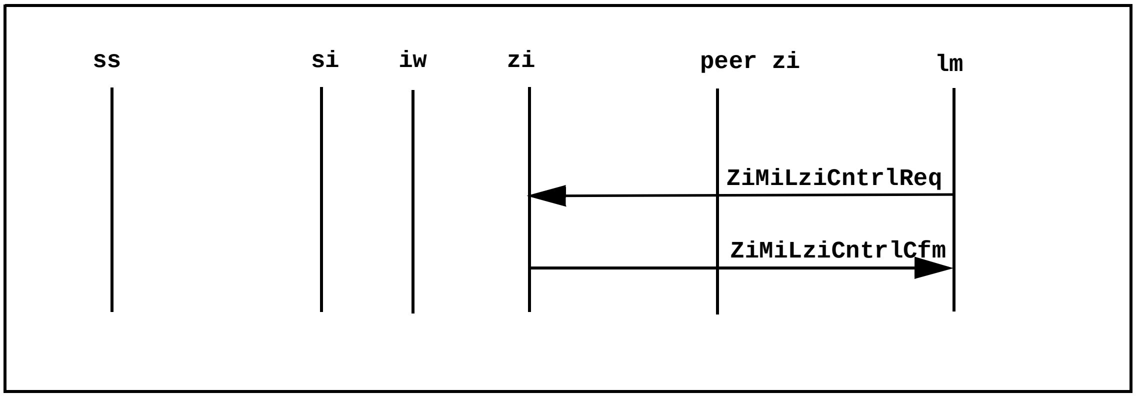


Figure 5-8: Data flow—control procedure

Interfaces

Peer Interface- General

Data Transfer

Name	Description
ZiPiOubDatReq	Outbound data request
ZiPiInbDatReq	Inbound data request
ZiPiOubDatCfm	Outbound data confirm
ZiPiInbDatCfm	Inbound data confirm

Interfaces

Peer Interface - Concepts

- The peer interface defines the interface with the peer PSF - ISUP. No external entity is required to support this interface.

- The generic calling sequence of the peer interface primitives is **`ZiPiAAAXXXYYY(pst,...)`**, where **`AAA`** is **`Inb/Oub`** and **`XXXYYY`** is the abbreviated primitive name (for example, **`DatReq`**).

- Post structure - **`pst`**

  Since the peer interface is always loosely coupled, **`pst->selector`** is not used for resolving the primitive at this interface.

Interfaces

Peer Interface - Specific

Data Request

```
PUBLIC S16 ZiPiOubDatReq (pst, mBuf)
Pst    *pst;   /* post structure */
Buffer mBuf;   /* message buffer */

PUBLIC S16 ZiPiInbDatReq (pst, mBuf)
Pst    *pst;   /* post structure */
Buffer mBuf;   /* message buffer */
```

- **pst**

  Post structure. Used to route the primitive from the calling layer to the called layer.

- **mBuf** contains both single and multiple control blocks state update information.

- PSF - ISUP sends data requests via the **ZiPiOubDatReq** primitive and receives data requests via the **ZiPiInbDatReq** primitive.

- The data flow is:

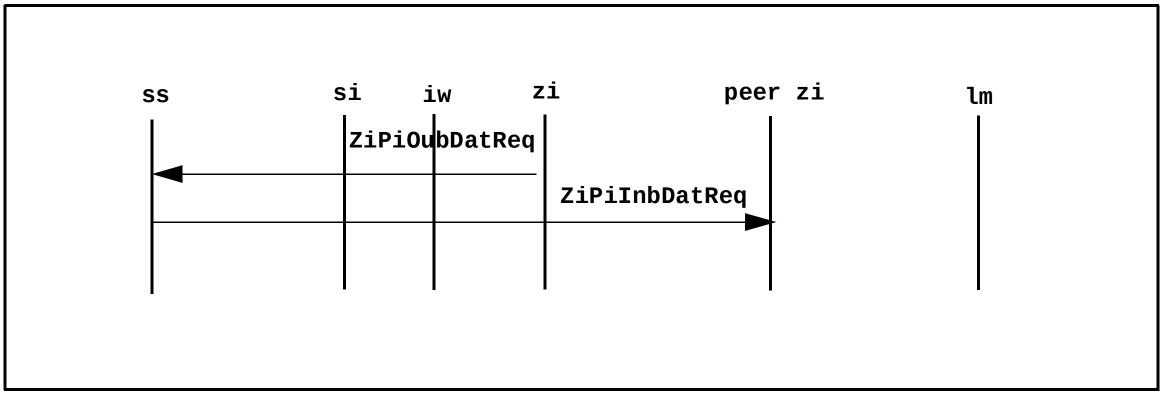


Figure 5-9: Data flow—data request

Interfaces

Peer Interface - Specific

Data Confirm

```
PUBLIC S16 ZiPiOubDatCfm (pst, status)
Pst *pst;      /* post structure */
U8  status;    /* confirm status */

PUBLIC S16 ZiPiInbDatCfm (pst, status)
Pst *pst;      /* post structure */
U8  status;    /* confirm status */
```

- **status**

  Data confirm status. Used by the standby PSF - ISUP to acknowledge the receipt of all bulk update messages from the active PSF - ISUP during warmstart or controlled switchover update.

  ```
  CMPFTHA_OK     /* ok status */
  CMPFTHA_NOK    /* not ok status */
  ```

- PSF - ISUP sends data confirms via the **ZiPiOubDatCfm** primitive and receives data confirms via the **ZiPiInbDatCfm** primitive.

- The data flow is:

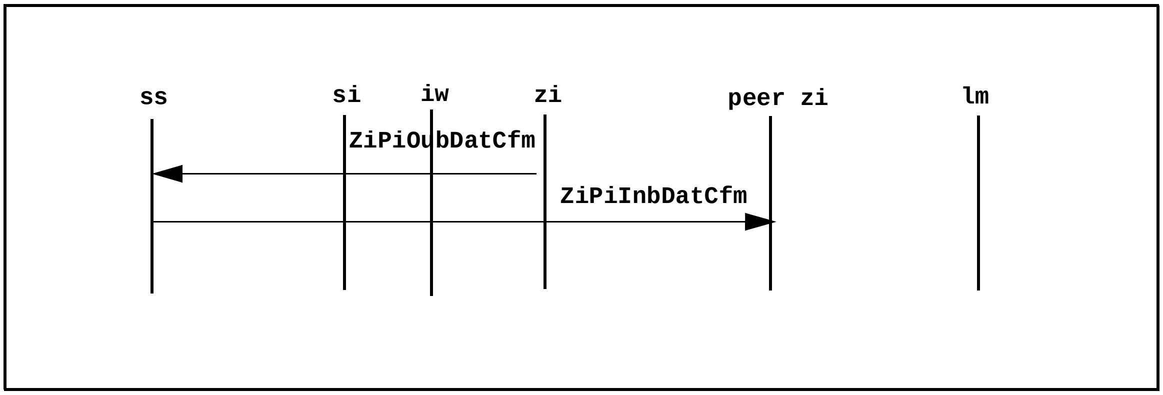


Figure 5-10: Data flow—data confirm

Interfaces

Protocol Layer Interface - Concepts

- Tightly coupled interface between the ISUP (PSIF - ISUP) protocol layer and PSF - ISUP through function calls and direct data structure access

- Internal interface only

- Active ISUP (PSIF - ISUP) uses this interface for run-time update of control blocks and for passing external events to PSF - ISUP (mostly through function calls).

- Standby PSF - ISUP uses this interface to update standby ISUP (PSIF - ISUP) control blocks with the update information received from the active PSF - ISUP (mostly through direct data structure access).

Interfaces

Protocol Layer Interface - Specific

Run-Time Update Procedure

- The run-time update procedure takes place when the active and the standby PSF - ISUP are operational and the active receives an external event that can change the stable states of the control blocks of ISUP (PSIF - ISUP).

- For ISUP, the data flow is:

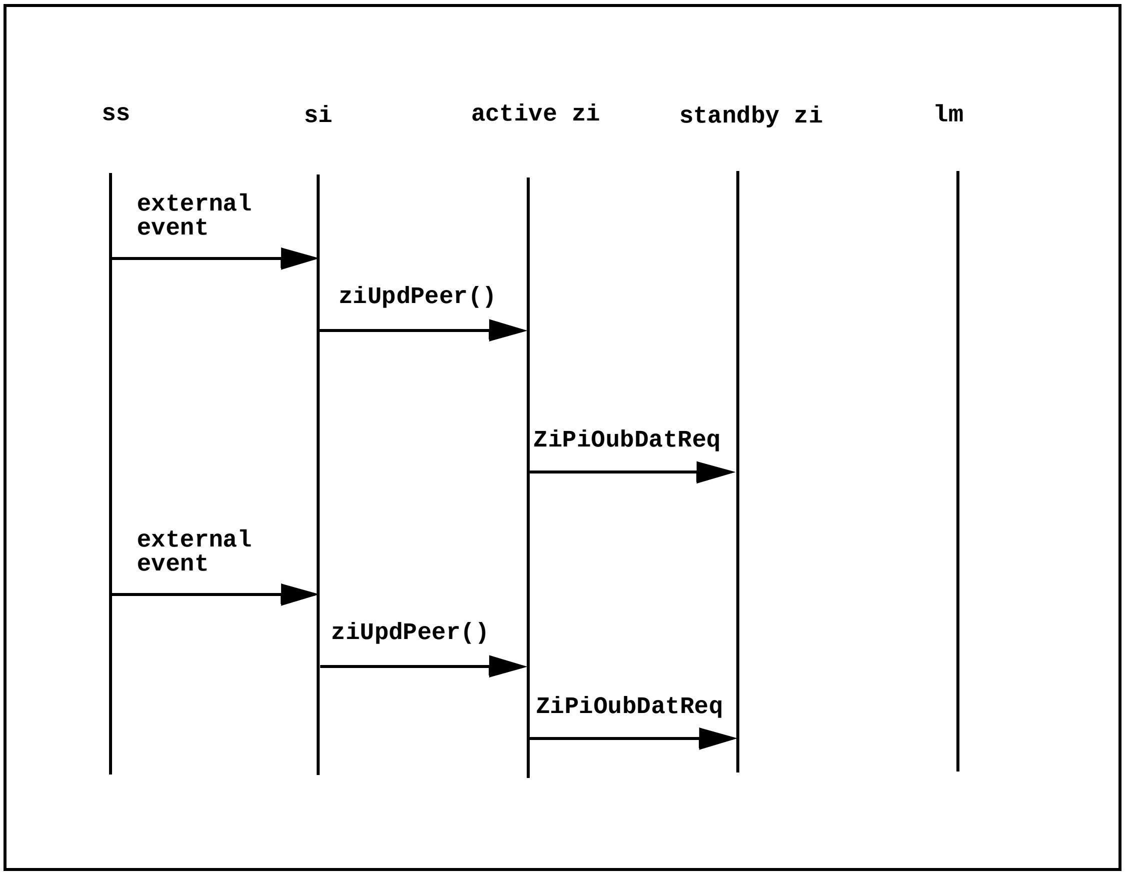


Figure 5-11: Data flow—run-time update procedure

Interfaces

Protocol Layer Interface - Specific

Run-Time Update Procedure

- For PSIF - ISUP, the data flow is:

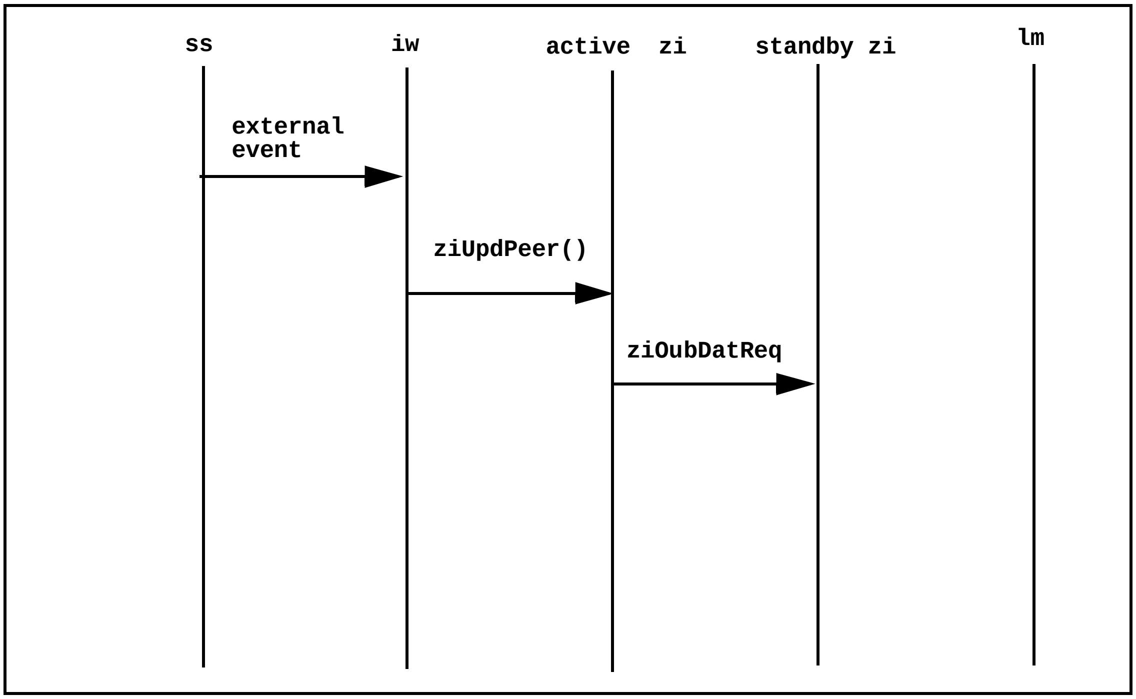


Figure 5-12: Data flow—run-time update procedure for PSIF - ISUP

Interfaces

Protocol Layer Interface - Specific

WarmStart Update Procedure

Successful Case

- The warmstart procedure takes place when the layer manager sends a control request (action **AWARMSTART**) to the active PSF - ISUP.

- The data flow is:

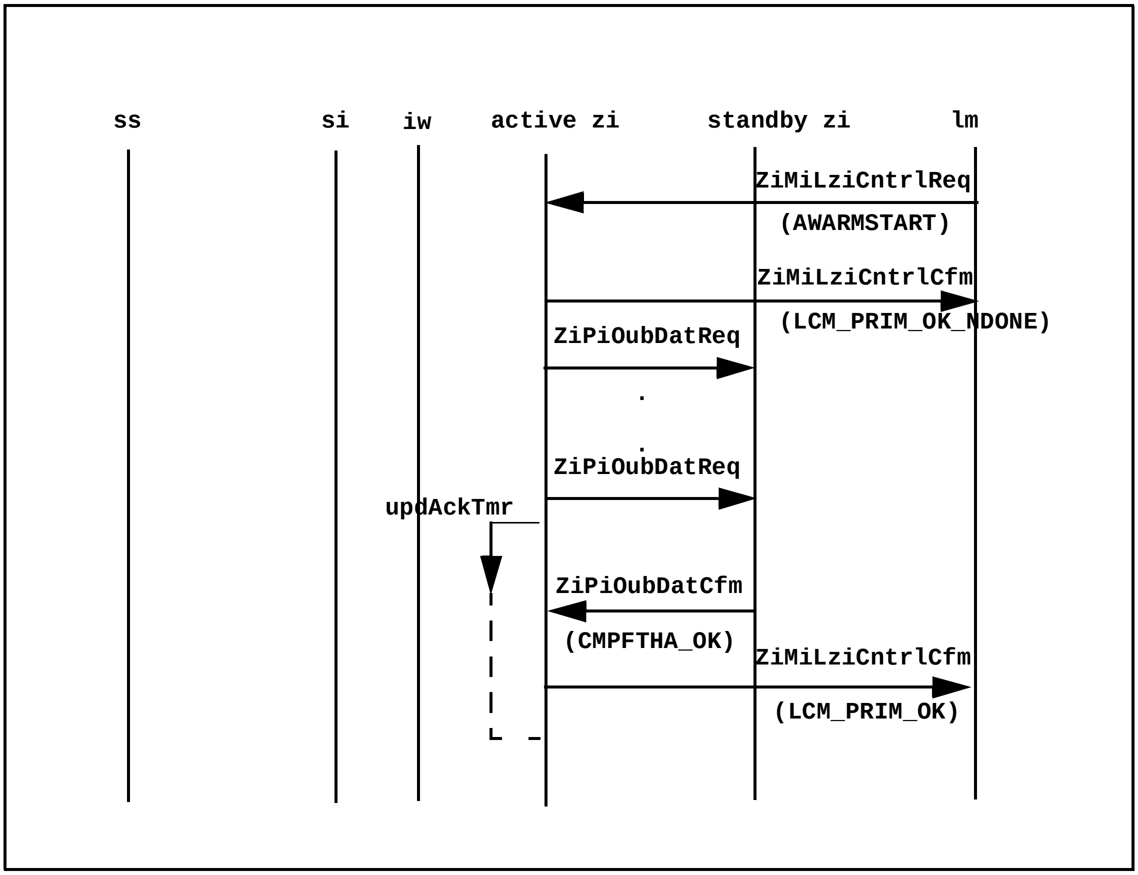


Figure 5-13: Data flow—warmstart update procedure, successful case

Interfaces

Protocol Layer Interface - Specific

WarmStart Update Procedure

Failure Case–Update Timer Expiry

- The data flow is:

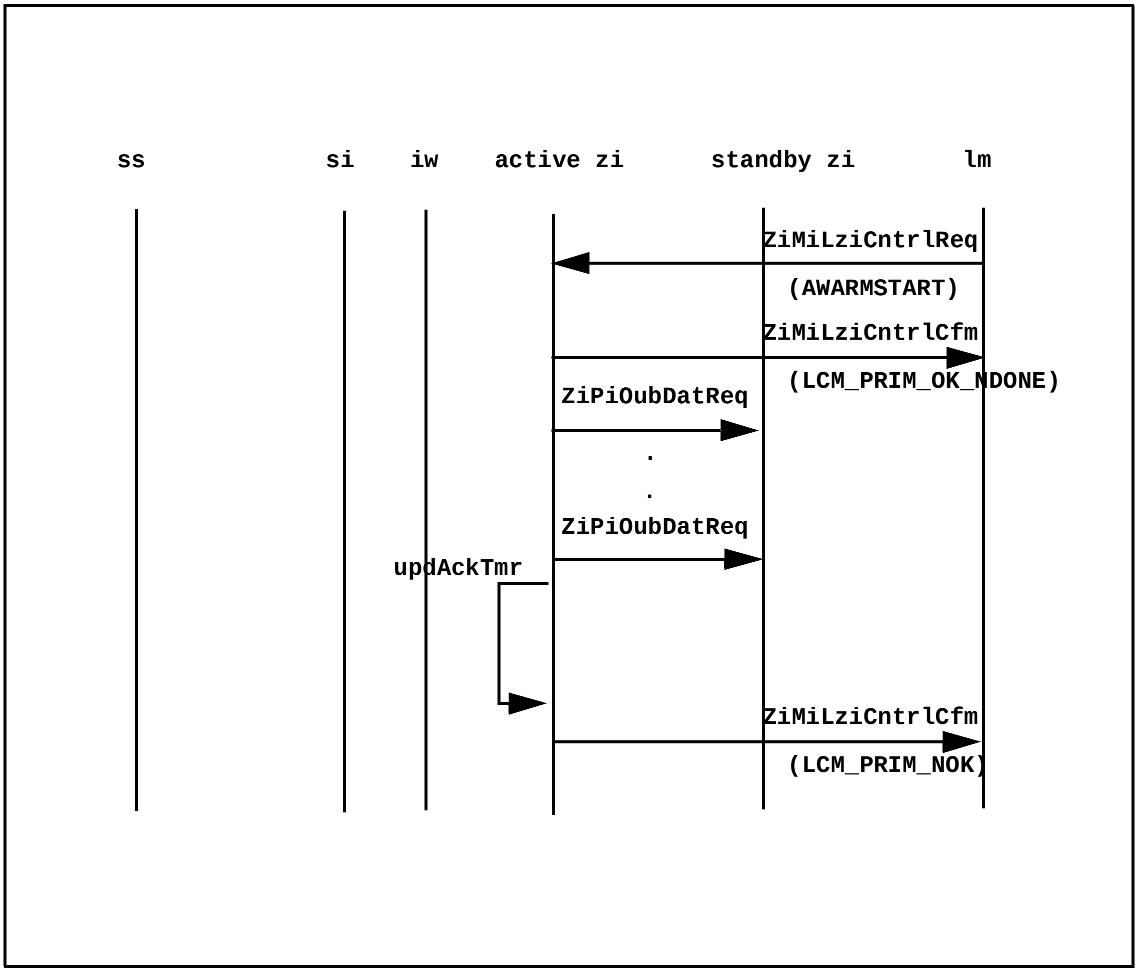


Figure 5-14: Data flow—warmstart update procedure, update timer expiry

Interfaces

Protocol Layer Interface - Specific

WarmStart Update Procedure

Failure Case–Negative Confirm from Standby

- The data flow is:

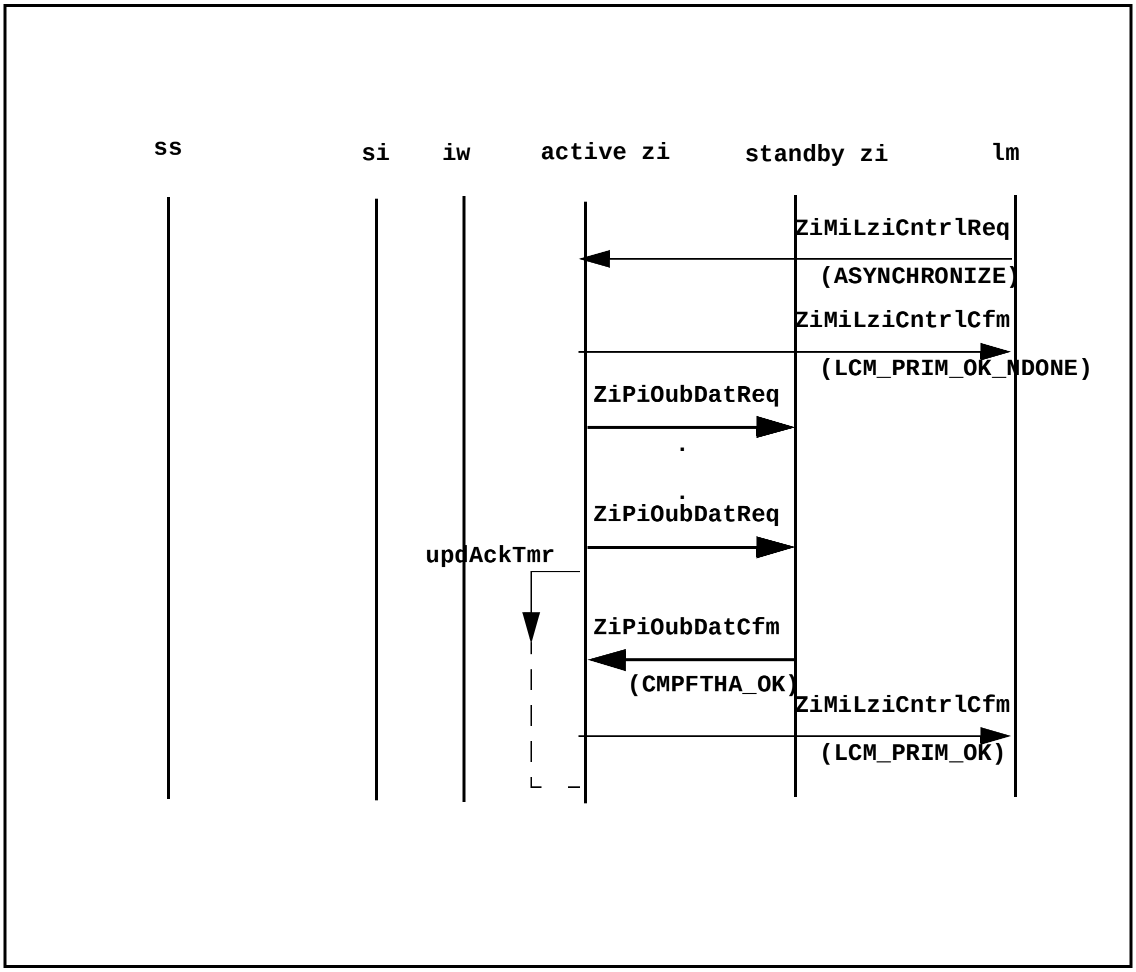


Figure 5-15: Data flow—warmstart update procedure, negative confirm from standby

Interfaces

Protocol Layer Interface - Specific

Synchronize Update Procedure

Successful Case

- The synchronize update procedure takes place when the layer manager sends a control request (action **ASYNCHRONIZE**) to the active PSF - ISUP.

- The data flow is:

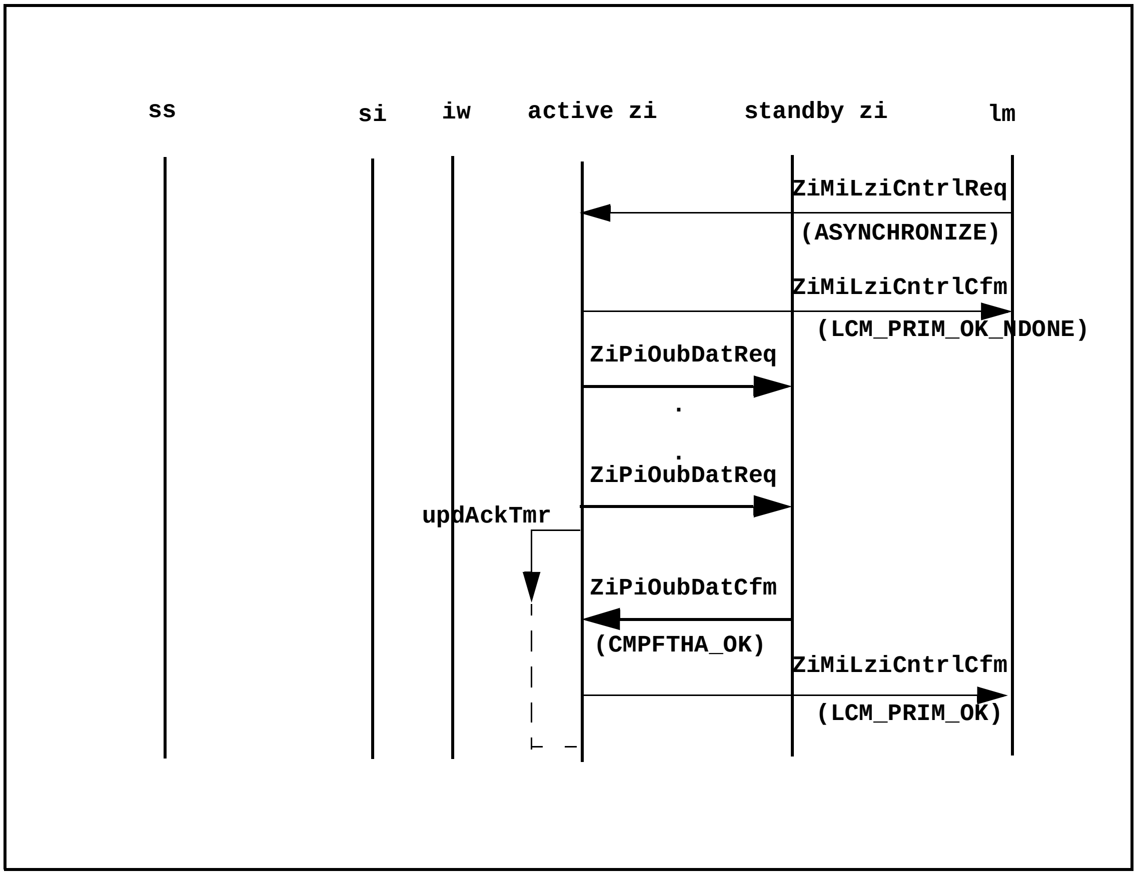


Figure 5-16: Data flow—synchronize update procedure, successful case

Interfaces

Protocol Layer Interface - Specific

Synchronize Update Procedure

Failure Case–Update Timer Expiry

- The data flow is:

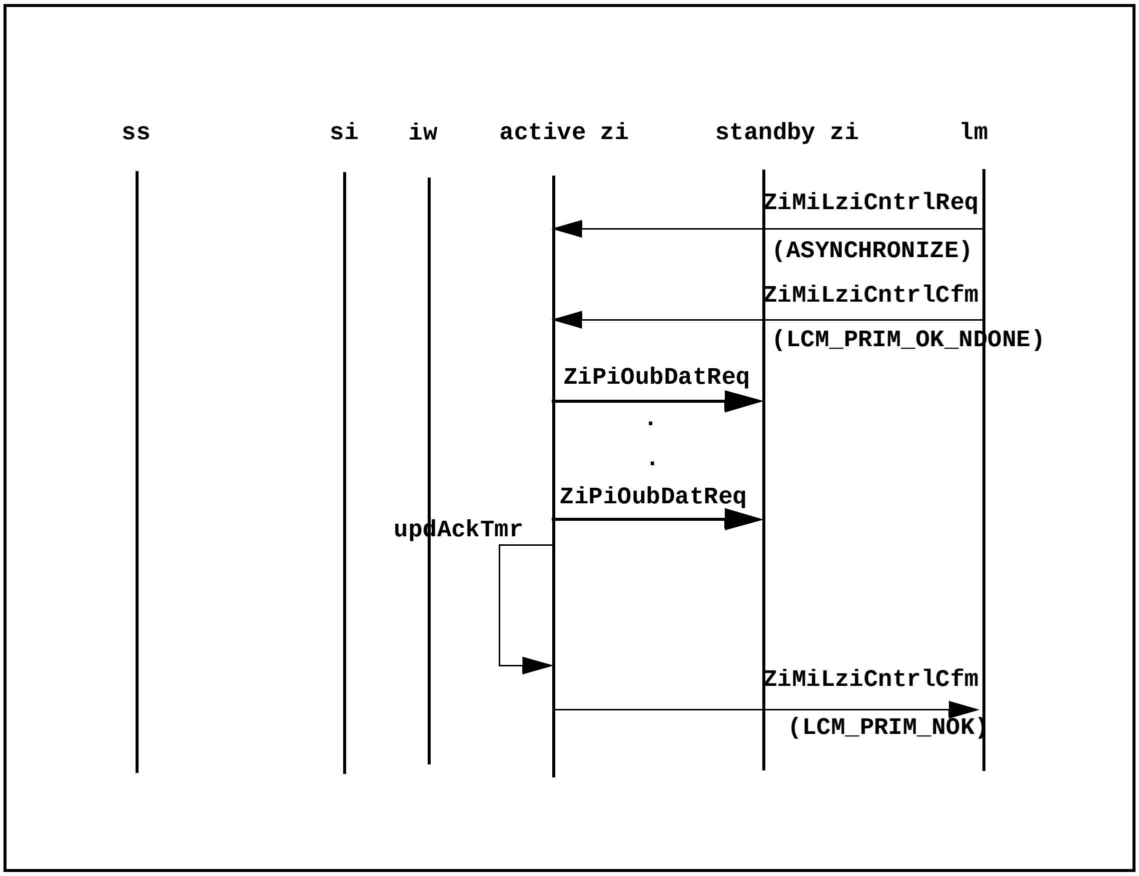


Figure 5-17: Data flow—synchronize update procedure, update timer expiry

Interfaces

Protocol Layer Interface - Specific

Synchronize Update Procedure

Failure Case–Negative Confirm from Standby

- The data flow is:

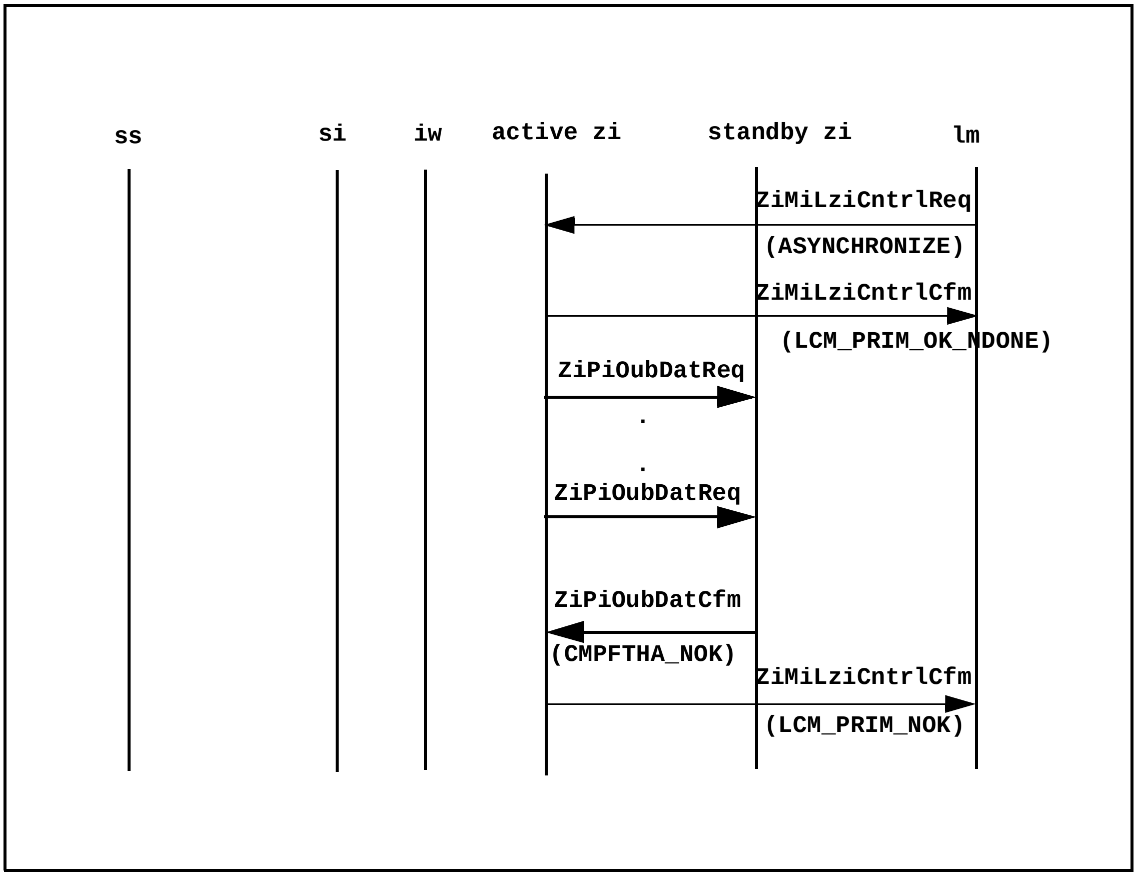


Figure 5-18: Data flow—synchronize update procedure, negative confirm from standby

6 INTERNAL ORGANIZATION

- Peer SAP

- State transition diagrams

Internal Organization

Peer SAP - Concepts

- The peer SAP contains information about the peer PSF - ISUP. The active PSF - ISUP uses this information to send update messages to the standby PSF - ISUP. The standby keeps track of the sequence of update messages coming from the active PSF - ISUP. PSF - ISUP also uses the information contained in the peer SAP to send data confirms.

- Single peer SAP

- The peer SAP exists as a global variable in PSF - ISUP. Memory for the peer SAP is allocated at compile-time rather than at configuration time.

Internal Organization

Peer SAP - Structure

```
typedef struct ziPeerSapCb
{
   Pst pst;                   /* Post structure */
   U8 state;                  /* sap state */
   U8 updState;               /* Update state */
   U8 rtNextUpdType;          /* run time update last table */
   U8 bulkNextUpdType;        /* last updated table type - buld update */
   U16 rtUpdSeqNum;           /* run time state update sequence number */
   U16 bulkUpdSeqNmb;         /* bulk update sequence number */
   U32 rtLastUpdIdx;          /* run time update index */
   U32 bulkLastUpdIdx;        /* bulk update index */
   PTR lastUpdHlEnt;          /* last updated hash list entry */
   U32 maxUpdMsgSize;         /* maximum update message size */
   Queue rtUpdHoldQ;          /* run time hold queue */
   S16 timeRes;               /* timer resolution */
   TmrCfg tUpdCompAck;        /* Update timer */
   CmTimer timer[ZIMAXTIMER]; /* timer structure */
   Pst defLmPst;              /* deferred layer management post */
   Bool cfgDone;              /* sap configuration done flag */
   TranId transId;            /* transaction Id */
} ZiPeerSapCb;
```

Internal Organization

State Transition - bndState

- **state** can have the following values:

```
CMPFTHA_BND              /* bound state */
CMPFTHA_UBND             /* unbound state */
```

- State transition (successful case):

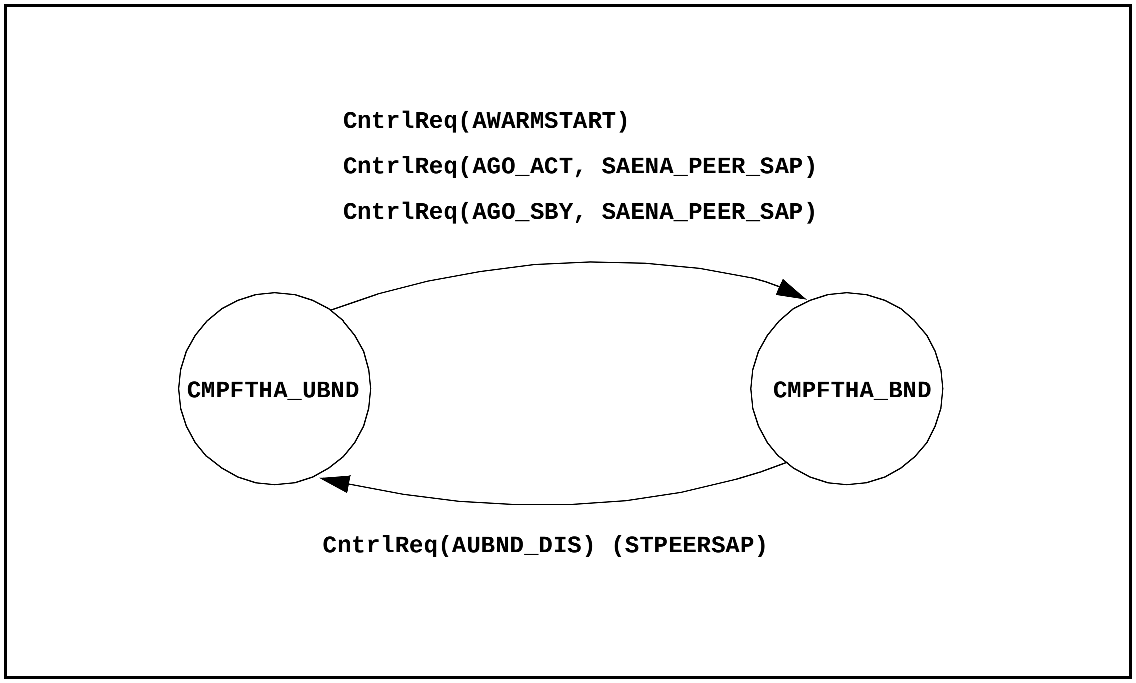

CntrlReq(AWARMSTART)
CntrlReq(AGO_ACT, SAENA_PEER_SAP)
CntrlReq(AGO_SBY, SAENA_PEER_SAP)
CMPFTHA_UBND
CMPFTHA_BND
CntrlReq(AUBND_DIS) (STPEERSAP)

Figure 6-1: State transition diagram—bind state

Internal Organization

State Transition - updState

- **`updState`** can have the following values:

```
CMPFTHA_IDLE           /* peer SAP in idle state */
CMPFTHA_WRMSTRT        /* peer SAP in warmstart state */
CMPFTHA_SYNC           /* peer SAP in synchronization state */
CMPFTHA_WRMSTRT_WAIT   /* waiting for warmstart data confirm */
CMPFTHA_SYNC_WAIT      /* waiting for sync data confirm  */
```

- State transition (successful case):

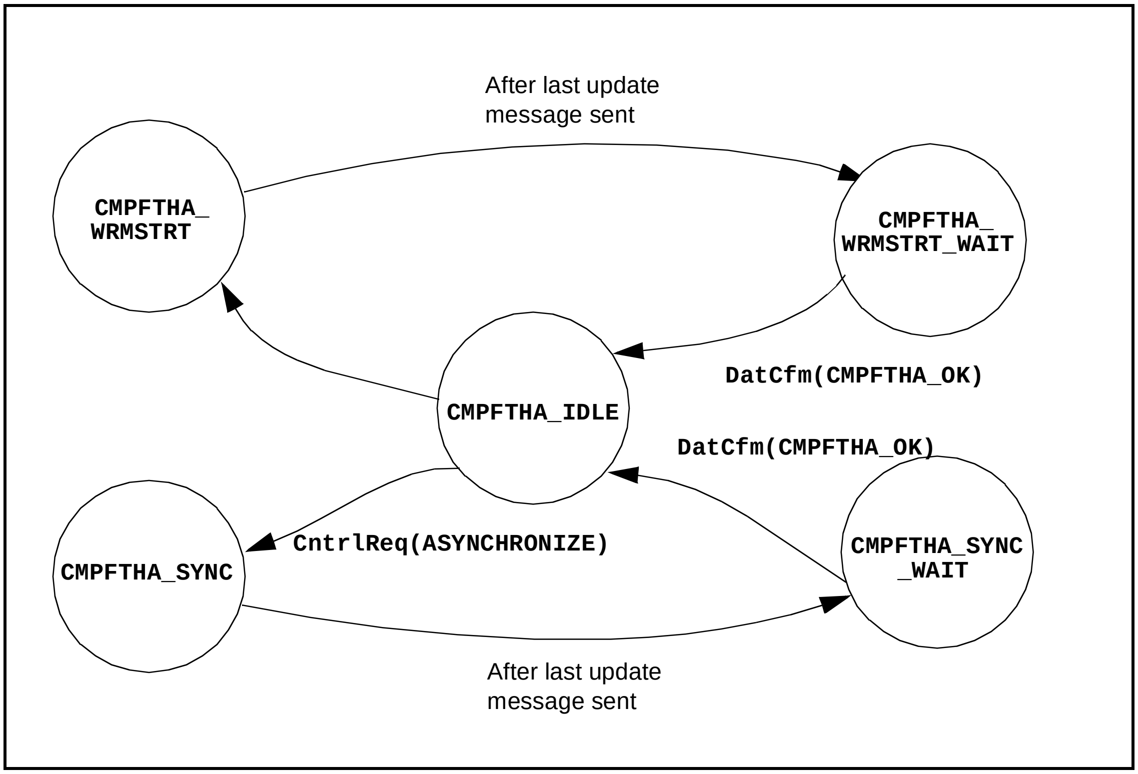


Figure 6-2: State transition diagram—update state

7 PORTATION

The steps for porting the PSF - ISUP software are:

1. Compile the PSF - ISUP software with ISUP (PSIF – ISUP). The PSF - ISUP software does not operate with ISUP releases before version 2.14 and PSIF – ISUP releases before version 1.3.
2. Compile the ISUP (PSIF – ISUP) protocol layer with the **`ZI`** compilation flag enabled
3. Compile PSF – ISUP with **`IW`** compile option for FT / HA support in PSIF – ISUP
4. Compile PSF – ISUP with **`SI_ACNT`** for accounting / billing support in ISUP
5. Supply ISUP's upper layer interface based on the SIT interface described in the *SIT Interface Service Definition*
6. Supply PSIF – ISUP's upper layer interface based on the CCT interface described in the *CCT Interface Service Definition*
7. Supply ISUP's lower layer interface based on the SNT interface described in the *SNT Interface Service Definition*
8. Supply ISUP's layer manager interface based on the LSI interface described in the *ISUP Service Definition*
9. Supply PSIF – ISUP's lower layer interface based on the RMT interface described in the *RMT Interface Service Definition*
10. Modify the **`makefile`** supplied with the PSF - ISUP software
11. Modify data types, as required, in the environment-dependent file (**`envdep.h`**)
12. Modify the environment options file (**`envopt.h`**)
13. Supply the system services interface using supplied prototype functions. This step is not necessary if linking directly to Trillium *Multi-threaded System Services (MTSS).*
14. Supply layer manager interface using supplied prototype functions
15. Link with any other existing software
16. Run the **`makefile`**