radisys

LTE TotaleNodeB Common Platform

High Level Design

3901464 1.0

Contents

Figures

Tables

1. Introduction

1.1. Purpose

This document describes various system software components categorized as part of the Radisys TotaleNodeB platforms.

1.2. Scope

This document details the component level decomposition and design details of components, communication between the components and interfaces with the TotaleNodeB non platform components.

1.3. Overview

Current version of TotaleNodeB aims to enable development of Home eNodeB (HeNB) products compliant to 3GPP Release 9.

3GPP defines the Home eNodeB Management System (HeMS) as an entity residing in the operator core network, which manages deployed HeNBs via the TR069 protocol defined by the Broadband Forum.

The layered architecture of the HeNB comprises of various Radisys components and third party components. Radisys components provide control plane and user plane LTE protocol stacks and HeNB specific functions.

Platform specific components support Operations, Administration, Maintenance and Performance measurements (OAM&P) functionalities on the HeNB. These include OAM exchanges with the HeMS and enables remote provisioning, monitoring and maintenance by the operator from the HeMS.

Platform components implements the following requirements governed by 3GPP specifications:

- Configuration Management
- Performance Management
- Fault Management
- Security Management

In addition to supporting remote management via the HeMS in deployment, the OAM component in TotaleNodeB supports the following using the TAPA Layer Manager interface:

- TotaleNodeB software initialization
- TotaleNodeB software configuration
- TotaleNodeB diagnostics

TotaleNodeB OAM is not a stand-alone product by itself, but is an essential component of the

TotaleNodeB smallcell software from Radisys.

For flexibility to customers, the OAM functionality in TotaleNodeB is split into two software components:

1. *OAM Agent*: This represents the TR069-compliant interactions of HENB with the HeMS. Individual customers may opt to replace this component from Radisys by software of their own choice.
2. *Software Management Entity*: This represents the implementation of all the device and software management functions in TotaleNodeB. The design for this entity must be flexible to allow customers to use the OAM Agent of their choice.

1.4. Abbreviations

The following table lists the abbreviation used in this document.

Acronym	Description
OAM	Operation, Administration and Maintenance
CWMP	Common WAN Management Protocol
CM	Configuration Management
FM	Fault Management
PM	Performance Management
HeMS	Home eNodeB Management System
CLI	Command Line Interface
KPI	Key Performance Indicators
RAN	Radio Access Network
REM	Radio Environment Map
RRM	Radio Resource Management
SSI	System Services Interface
IPsec	IP Security
SON	Self-Organizing Networks
MIB	Management Information Base
NTP	Network Time Protocol
FTP	File Transfer Protocol
TCP	Transmission Control Protocol

TCP/IP	Transmission Control Protocol / Internet Protocol
UDP	User Datagram Protocol
PHY	Physical Layer
FSM	Finite State Machine
NAS	Non-Access-Stratum

1.5. Terminologies Used

Terms	Meaning
HeNB (Home eNodeB)	The common platform and the eNB App together is called as HeNB.
OAM-Messenger	OAM-Messenger is the interface between OAM and Stack Manager.
SMM	Interface between OAM-Messenger and Stack Manager.

1.6. References

1.6.1 Standards

1. [TR196], “Femto Access Point Service Data Model”, Issue 2, November 2011.
2. [TR069], “CPE WAN Management Protocol”, Issue 1, Amendment 4, July 2011.

1.6.2 Shared Documents

None.

1.7. Release History

The following table lists the history of changes in successive revisions of this document:

Version	Date	Author (s)	Description
1.0	January 21, 2013	Chiranjeevi, Suhas	Conforms to TotaleNodeB Solution release, version 1.0.
0.1	November 15, 2012	Platform Team	Initial Draft Version

1.8. Proprietary Documents

1 Radisys_TeNB_Software_Architecture

2 [FSRS Config Management], System requirements, Configuration Management in TeNB, July 2012

3 [FSRS Performance Management], System requirements, Performance Management in TeNB, July 2012

4 [FSRS Fault Management], System requirements, Performance Management in TeNB, Jun 2012

5 [Data Model], Radisys_Data_Model

2. Functional Description

2.1. Common Platforms Infrastructure

This section describes the different infrastructures used by the common platform.

Following infrastructures used in the common platform:

1. Messaging Framework
2. CLI
3. Logging

2.2. Messaging Framework

Post Office is used as a messaging framework between the different Radisys Platform components. Post Office is a process by itself and provides services to its users. Logical messaging entities (namely, OAM, TR069, and so on) deliver messages by UDP datagram to each other. The Post Office framework allows the entities to communicate independent of the process which implements them and the IP host and UDP port where they are terminated.

Entities (software components) use messaging to provide services to other entities and/or to use the services provided by other entities. The characteristics of the Post Office messaging framework are listed as follows:

1. Each entity is assigned a numeric identity.
2. A ‘static entity’ is an entity that has a well-known identity assigned to it.
3. The identities of all static entities are globally unique within the messaging system.
4. Each application supports zero or more messaging "entities" where an entity represents an addressable message source/destination.
5. Each message is a C++ class derived from the Serialisable header file (libs/common/include/system/Serialisable.h). Anything that serialise/deserialise itself to/from a byte stream.
6. Messages are exchanged through UDP packets consisting of the serialised data and an addressing header.
7. Each application registers its entities by sending a message to the POSTOFFICE entity.

8. The post office entity (encapsulated in the post-office application) maintains a routing table that it distributes to each entity so that every entity knows where every other entity resides (IP address and port).
9. Most messaging is connection-less, but connection-oriented messaging is also supported.

To deliver these services, the post office:

1. Provides a static entity hosted at a well-known endpoint.
2. The post office's entity is assigned the globally unique identity '1' (one).
3. Every other entity in the system is configured with an IP address.
4. The post office endpoint is assigned a well-known UDP port number, 6000.
5. Allows other applications to register their static entities by means of sending a message to the post office from the hosting endpoint.
6. Distributes the 'static entity to endpoint' map, to all endpoints which have registered one or more static entities.
7. Forwards messages with a static 'From' entity without re-writing the 'From' entity.

2.2.1. Sequence Diagram

The following sequence diagram describes the messaging entity register sequence diagram.

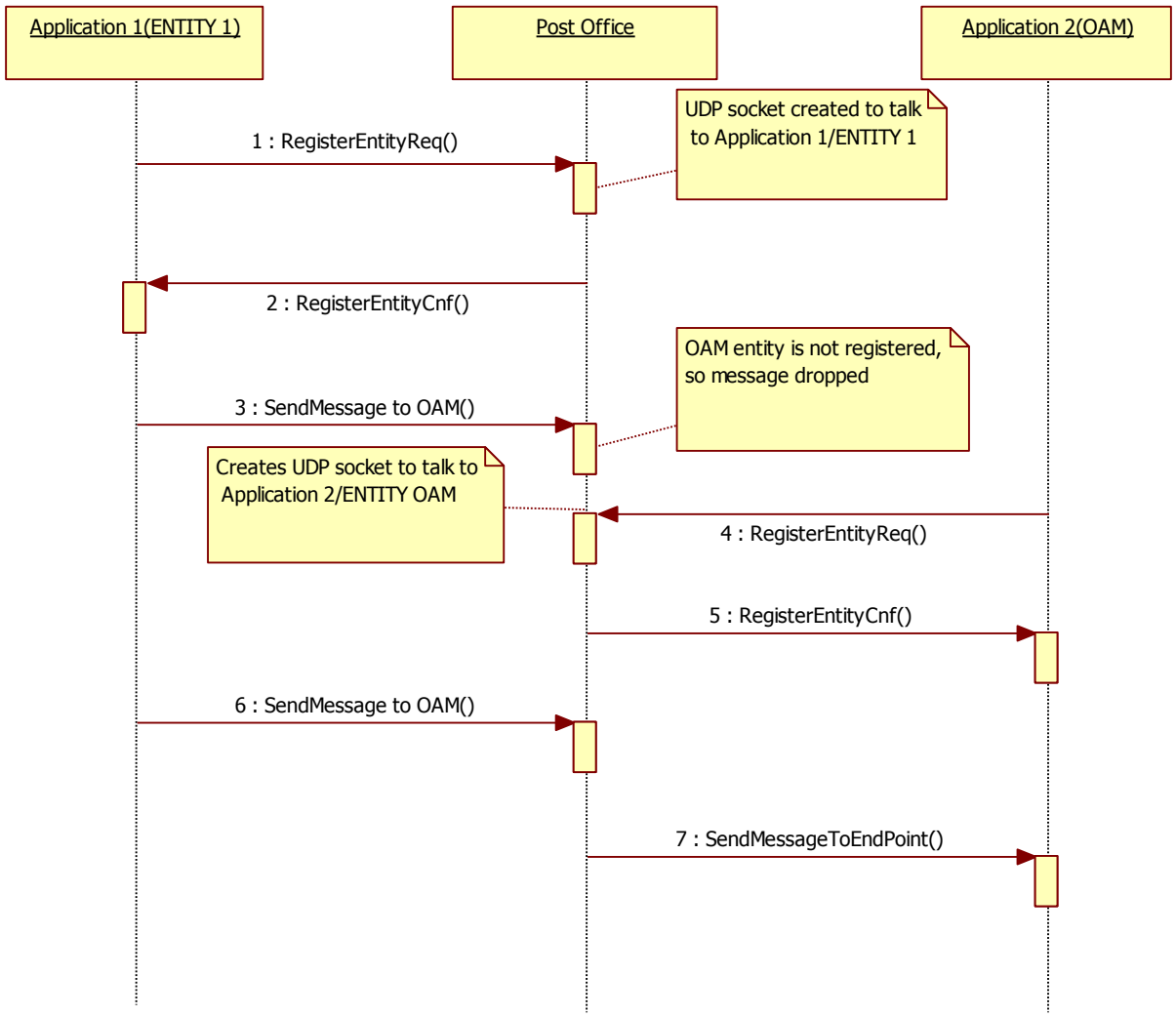

Figure 1: Messaging Entity Register Sequence Diagram

1. ENTITY 1 of application 1 calls RegisterEntityReq (ENTITY 1) to get registered with Post Office.
2. After receiving registration request, Post Office creates UDP socket (Port number needs to be configured against each entity in *nas-configuration-system file*) for the application and sends the Registration Confirmation.
3. ENTITY 1 tries to send a message to OAM, but Post Office drops the message as OAM has an entity which is not yet registered with Post Office.
4. Entity OAM of Application 2 sends the Registration request to Post Office.

5. After receiving registration request, Post Office creates UDP socket for the application and sends the registration confirmation.
6. ENTITY 1 tries to send a message to OAM by calling sendMessage().
7. Post Office forwards the message to the OAM.

2.2.2. Class Diagram

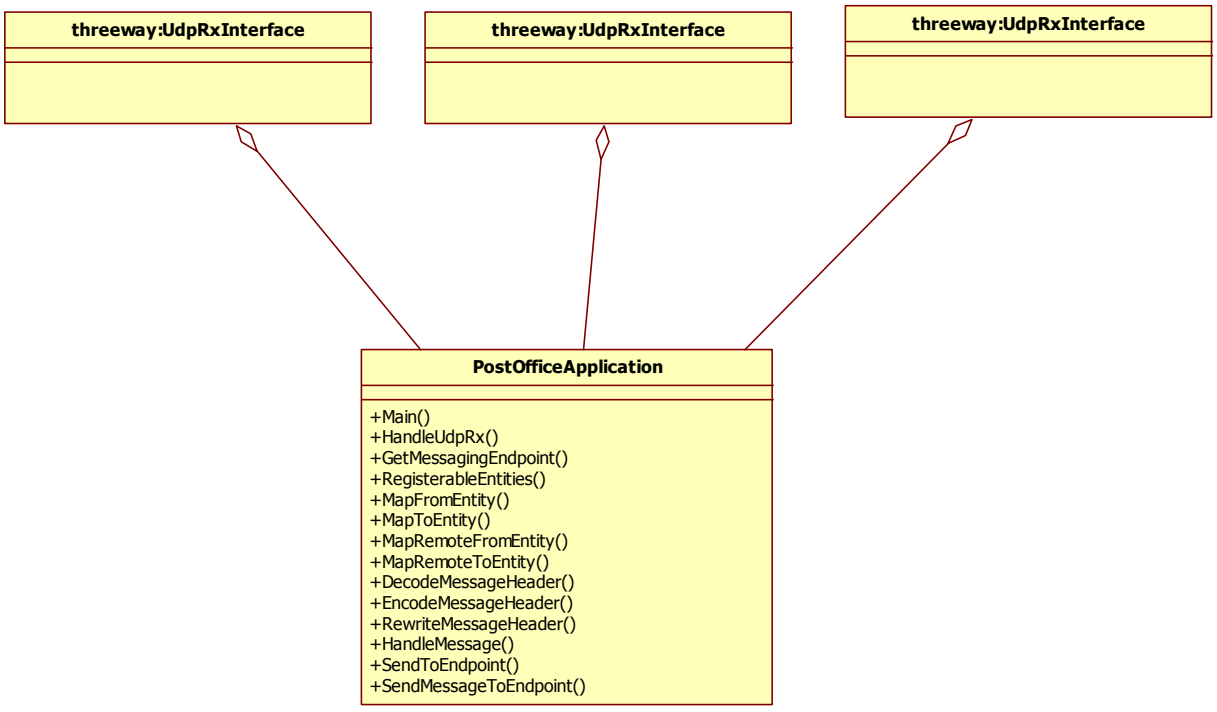

Figure 2: Class Diagram for Post Office

2.2.3. Message format

This section describes the message format used by the Post Office framework

Table 1: Generic Message Format

Bytes	Field	Notes
0..3	From Entity	In network byte order
4..7	To Entity	In network byte order
8..11	Message ID	In network byte order
12..end	Message Specific Data	Optional

Following are the some examples of serialized messages as they appear in a UDP packet. This is the UDP payload. All values are big endian.

RegisterReq

Table 2: Register Request Message Format

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
fromEntity				toEntity				serializationId(REGISTER_REQ)				Serialized Data			

MessageRoutingInd

Table 3: Message Routing Indication Message format

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	..	N
fromEntity				toEntity				serializationId (MESSAGE_ROUTING_IND)				Serialized Data														
												routingTableLen				routingTable										
																D	N	entry							entry	
																		E	IP address				Port		.....	

D => 0 = No default route, 1 = First entry is default route

N => Number of entries

E => Entity

MessagingConnectionReq (contains an embedded message)

Table 4: Message Connection Request Message Format

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>..</th><th>N</th></tr>
<tr><td colspan="4">fromEntity</td><td colspan="4">toEntity</td><td colspan="4">serializationId(MESSAGE_CONNECTION_REQ)</td><td colspan="15">Serialized Data</td></tr>
<tr><td colspan="12"></td><td colspan="4">sourceConnId</td><td colspan="11">Embedded msg</td></tr>
<tr><td colspan="16"></td><td colspan="4">SerializationId</td><td colspan="4">Length of Data</td><td colspan="3">Data</td></tr>
</table>

2.2.4. Interfaces

The following table describes the external interfaces provided by Post Office.

Table 5: Interfaces Provided by Post Office

Interface	Description
RegisterEntityReq	Request to register the entity in Post Office. Entity ID is used as the parameters.
SendMessage	Send message to another entity. Parameters used are messagetosend, to entity id, from entity id.

2.2.5. Files Used/Added/Modified

The post office files are found in the following mentioned location.

tenb_commonplatform/software/apps/fap/post-office

Table 6: Files in Post Office

File Name
PostOfficeApplication.cpp
PostOfficeApplication.h

Table 7: Dependency Files for Post Office

File Name
system/Application.h
PostOfficeApplication.h
system/Serialisable.h
messaging/transport/MessagingEntity.h
messaging/transport/Messenger.h
messaging/transport/MessageRoutingTable.h
messaging/transport/MessageRouteEntry.h
Configfiles/nas-system-configuration

2.3. Command Line Interface (CLI)

CLI is another infrastructure provided by the Radisys Common Platform. CLI is standalone application (system process) that can be started/stopped at any time.

The CLI application provides a command prompt where the user can:

- query available commands (aka help)
- enter commands and receive responses

It must be noted that though CLI is useful in many ways it must be used restrictively. The main interface for configuring the system is via TR069 and OAM, which is primarily used by the HeMS for configuring the HeNB.

CLI run in two modes as follows:

a) Interactive mode

In this mode, CLI application accepts input and allows commands to be constructed on its own command line with line editing, command history and command completion.

b) Non-interactive mode

This mode is intended to allow CLI commands to be invoked form shell scripts and so on. The command to be run is specified in “one-shot” and the application returns the status value returned by CLI command executed. Negative values indicate an error during execution.

2.4. Design Details

2.4.1. State Machine

The figure and table describes the state machine used in CLI as follows.

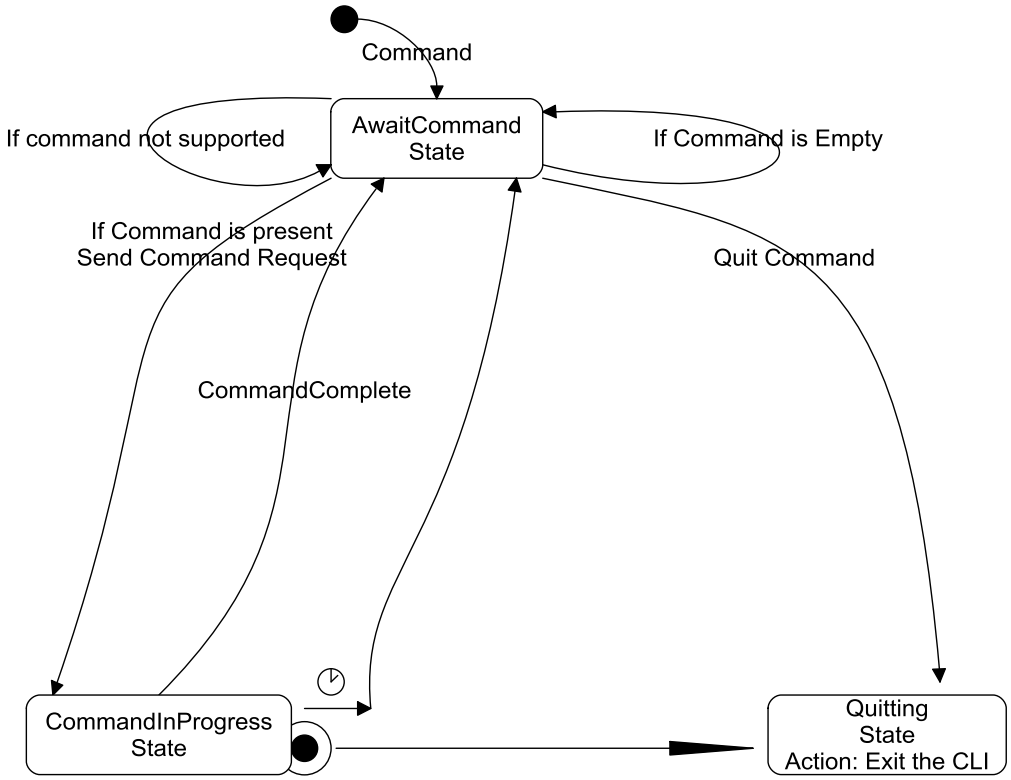

Figure 3: CLI State Transition Diagram

Table 8: CLI State Machine list

State	Event	Action	New State
AwaitCommand	CommandLine	If QueryQuitCommand, then Quit	Quitting
		If QueryCommand is Empty	AwaitCommand
		If CommandLineParser SendCommandReq	CommandInProgress
		If none of the above, then Reject the command	AwaitCommand
CommandInProgress	Timeout	FeedbackTimeout	AwaitCommand
	CommandComplete	FeedbackCompletionResult	AwaitCommand
	Stop	FeedbackStopReason	Quitting
	CommandResponse	FeedbackResponse	CommandInProgress
Quitting		Exits from the CLI	

2.4.2. Class Diagram

Figure 4: CLI Class Diagram

2.4.3. Files Used/Added/Modified

Path: *tenb_commonplatform/software/apps/utilities/cli*

CliUserInput.h
CliUserInput.cpp
CliTypes.h
CliInput.h
CliInput.cpp
CliFsmTypes.h
CliFsmInterfaceStandard.h

CliFsmInterfaceStandard.cpp
CliFsmInterface.h
Cli_fsm_interface.h
CliFsmInterface.cpp
CliFsmInterfaceCanonical.h
CliFsmInterfaceCanonical.cpp
CliFsmInterfaceAte.h
CliFsmInterfaceAte.cpp
CliFsm.h
Cli_fsm.h
CliFsm.cpp
Cli_fsm.cpp
Cli.fsm ***CLI FSM***
CliEvents.h
CliEvents.cpp
CliCommand.h
CliCommand.cpp
CliCmdFileMgr.h
CliCmdFile.h
CliCmdFile.cpp
CliApplication.h
CliApplication.cpp ***CLI Application***

2.5. Usage

1. CLI is used to retrieve the KPI counters by using the below mentioned command:
   oam.pollkpis <interval|total>
2. CLI is used to get/set the configuration parameters to OAM.
   For example:
   - *oam.getwild LTE_* - Retrieves all the LTE parameters
   - *oam.get <param_name>* - Retrieves the single parameter
   - *oam.set <param_name> <value>* - Set the parameter

2.6. Logging

Trace is the logging/debug feature that writes trace information to local FAP files and console and optionally to UDP to allow remote trace.

Trace supports functionality like:

- Each application specifies an explicit local file to write traces to.
- Trace files are located at a specific location (say /dbx/trace).
- Trace files have standard file format, which makes it easy to identify date-time, application which created the trace file.
- Log/trace files are rotated after they reach a pre-defined size (this is however configurable via the MIB).
- Trace macros are called by 'C' functions as well.

2.6.1. Design Details

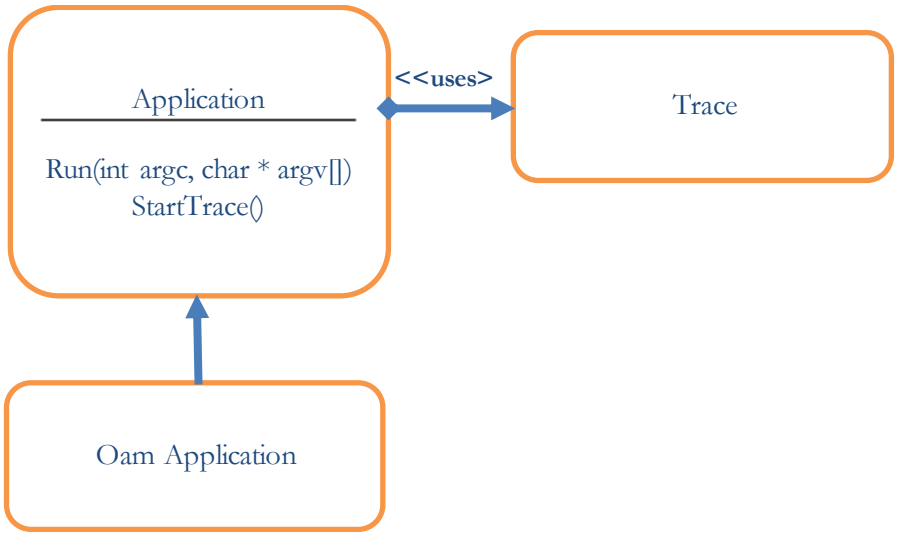

Figure 5: Example of Trace functionality used by OAM application

Class 'Trace' is defined in Trace.h file and is a singleton. Each application object uses one instance of 'Trace' object.

Each 'Application' starts with Application::Run () function which starts the Trace.

The 'Trace' object is initialized in Application::StartTrace () which stores:

- Application information,
- TraceLevelMask
- UDP Logging enable/disable state
- File logging parameters (like directory file name, file limit)

2.6.2. Files Used

Directory: *tenb_commonplatform/software/libs/common/system*

Files: *Trace.h*

Trace.cpp

2.6.3. Usage of Logging

2.6.3.1. Trace Macros

Tracing is achieved through macros that allow us to efficiently include/exclude trace and include additional info such as file/line.

All the macros automatically append a "\n" so you don't have to.

The macros are defined in:

#include <system/Trace.h>

The full set of trace macros are:

Macro	Purpose
ENTER	Use at start of all functions except simple set/get defined in header file. May be used to provide features like call stack tracing and stack depth checking.
EXIT	Use at end of all void functions except simple set/get defined in header file.
RETURN	Use at any function return point for non-void functions.
RETURN_VOID	Use at any function return point for void functions.

Macro	Purpose
TRACE_MSG	Trace just a message.
TRACE	Trace a message and a single variable of simple type.
TRACE_PRINTF	Trace in printf style.
TRACE_MSG_LEV TRACE_LEV TRACE_PRINTF_LEV	As TRACE* above but with a criticality level.
TRACE_MSG_CAT TRACE_CAT TRACE_PRINTF_CAT	As TRACE*LEV above but with an additional user defined category.
TRACE_HEX	Trace a single u32 value as hex.
TRACE_HEX_MSG	Trace an array of u8 data as hex.

Macro	Purpose
MESSAGE_TAG MESSAGE_TAG_SHORT MESSAGE_TAG_LEV MESSAGE_TAG_SHORT_LEV MESSAGE_TAG_SHORT_CAT	Macros for generating Protocol Professor style trace.

Macro	Purpose
AIRV_ASSERT	Assert macro to use to capture coding bugs.
AIRV_ASSERT_PRINTF	As AIRV_ASSERT with additional printf style message.

2.6.3.2. Trace Levels and Categories

There are three methods of expressing the "level" of a trace statement:

Default Trace

Use the macros that do not take a level or category, for example, TRACE, TRACE_PRINTF. The criticality is assumed to be INFO and there is no associated category.

Turning INFO level on/off is the only way to control whether this trace is output or not.

Trace Criticality

Use the criticality levels along with a macro that accepts them for example,

```
TRACE_LEV (TRACE_VERBOSE, "Some verbose trace of a variable", myVar);
TRACE_PRINTF_LEV (TRACE_CRITICAL, "Uh oh, check this a=%lu, b=%s", myU32, myStr);
```

Trace Categories

User defined categories are used on the _CAT macros to express trace categories as per user convenience.

Trace categories are defined as a bit mask that is unique to a particular application. So for the l1l2 categories we have:

```
static const u32 TRACE_CAT_OAM  = 0x00000001;
```

These categories have to be registered at runtime:

```
Trace::GetInstance()->RegisterTraceCategory(TRACE_CAT_OAM, "OAM");
```

Then the _CAT macros are used as follows:

```
TRACE_CAT (TRACE_CAT_OAM, TRACE_VERBOSE, "Some verbose trace of OAM variable", myRlcVar);
```

Controlling Trace Level

Trace levels and categories are read and set via CLI:

```
<entity>.tracelev <level> <on|off>
<entity>.tracecat <category> <on|off>
```

2.7. Configuration Management

This section describes the different components and procedures involved in configuration management in common platform. Below are the various components involved:

1. Operation, Administration, Maintenance (OAM)
2. TR069 Client
3. OAM-Messenger

Each of these components communicates using Post Office messaging application.

2.8. Functional Description

The following diagram describes OAM module interaction with TR069 Client and OAM-Messenger.

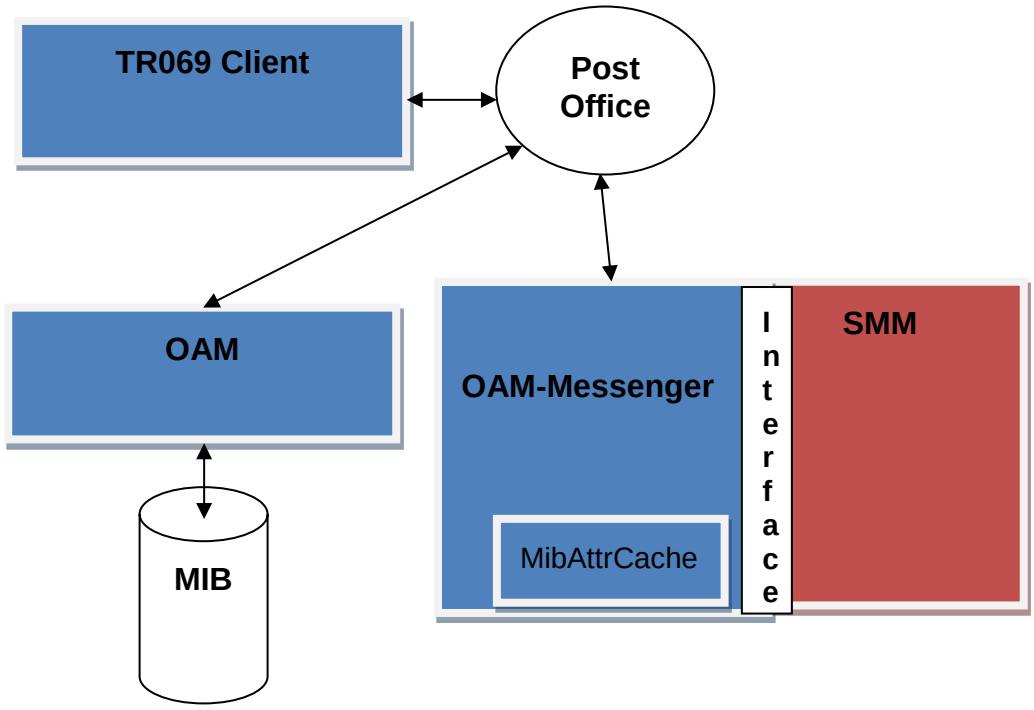

Figure 6: Interaction Diagram of OAM Module

2.8.1. Operation, Administration, Maintenance (OAM)

Operation, Administration, Maintenance (OAM) Subsystem is a key component and plays a vital role in managing the HeNB. OAM controls and acts upon the configuration of the HeNB which is stored in a central database called the Managed Information Base (MIB). OAM module is the central controller within the HeNB and drives the top level state and tasks carried out in the HeNB. The OAM drives the top level state machine for the HeNB and ensures that the eNodeB stack is not set up and able to transmit until the appropriate registration has obtained from the HeMS and S1 connection set up with MME/S-GW. There is a management interface between the OAM and eNodeB stack to control initialization, starting and stopping of the eNodeB stack.

The HeNB's management interface to the HeMS is driven by the TR069 unit in the HeNB specific functional block. This unit interacts with the OAM (via Post-Office) which is responsible for reacting to HeMS messaging by changing the state and/or configuration of the HeNB.

OAM manages the security aspects of the HNB's Ethernet backhaul connection. It is responsible for configuring and setting up the secure IP tunnel to the remainder of the operator's network infrastructure. OAM is responsible for various functionalities namely Configuration Management (CM), Performance Management (PM), Fault Management (FM) and Software Management. OAM along with TR069 CWMP protocol allows HeNB to be remotely managed by the HeNB Management System (HeMS).

2.8.1.1. Class Diagram

OAM Application Initialization procedure is described as follows:

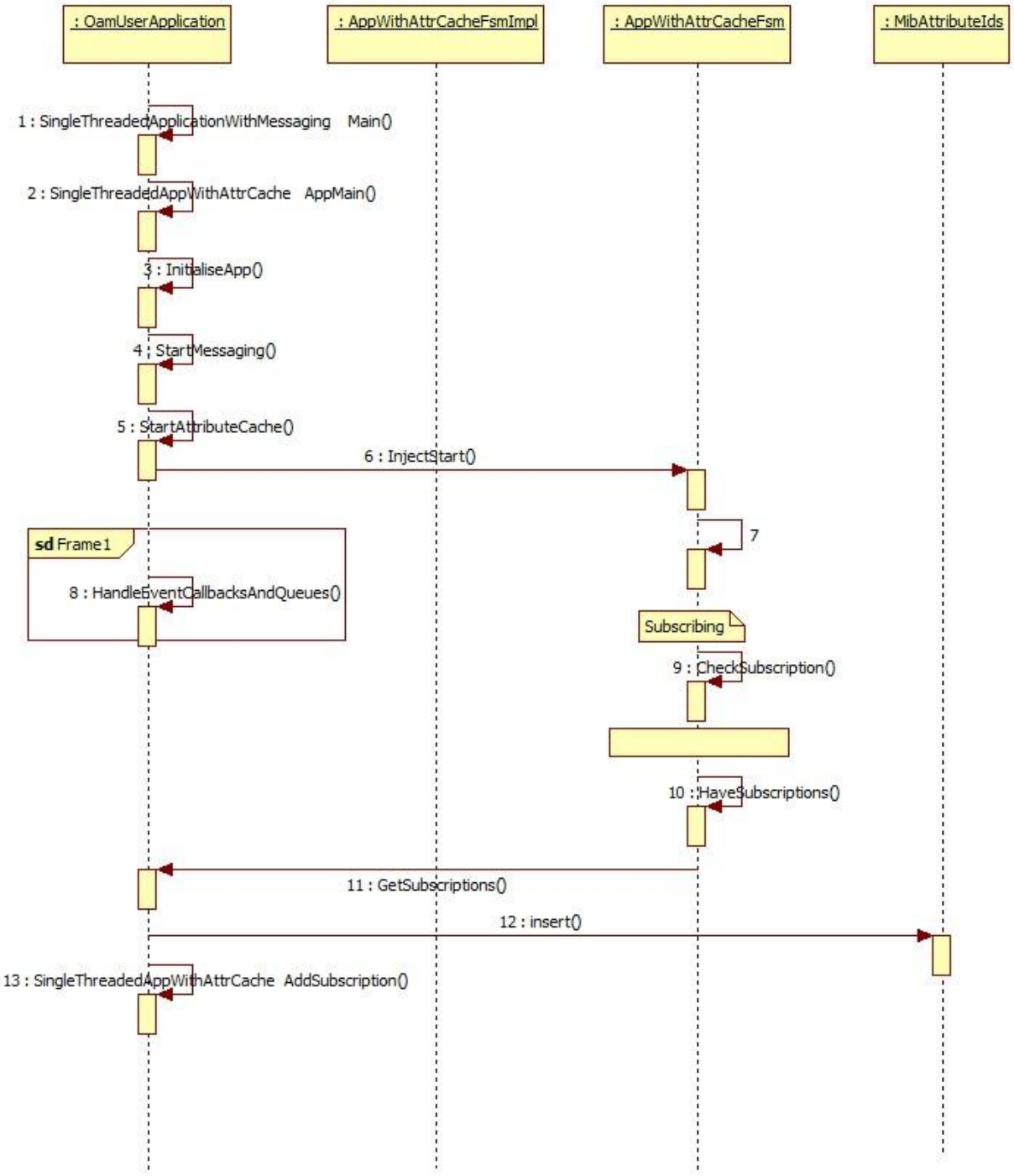

Figure 7: Class Diagram

The following is the class diagram of OAM module:

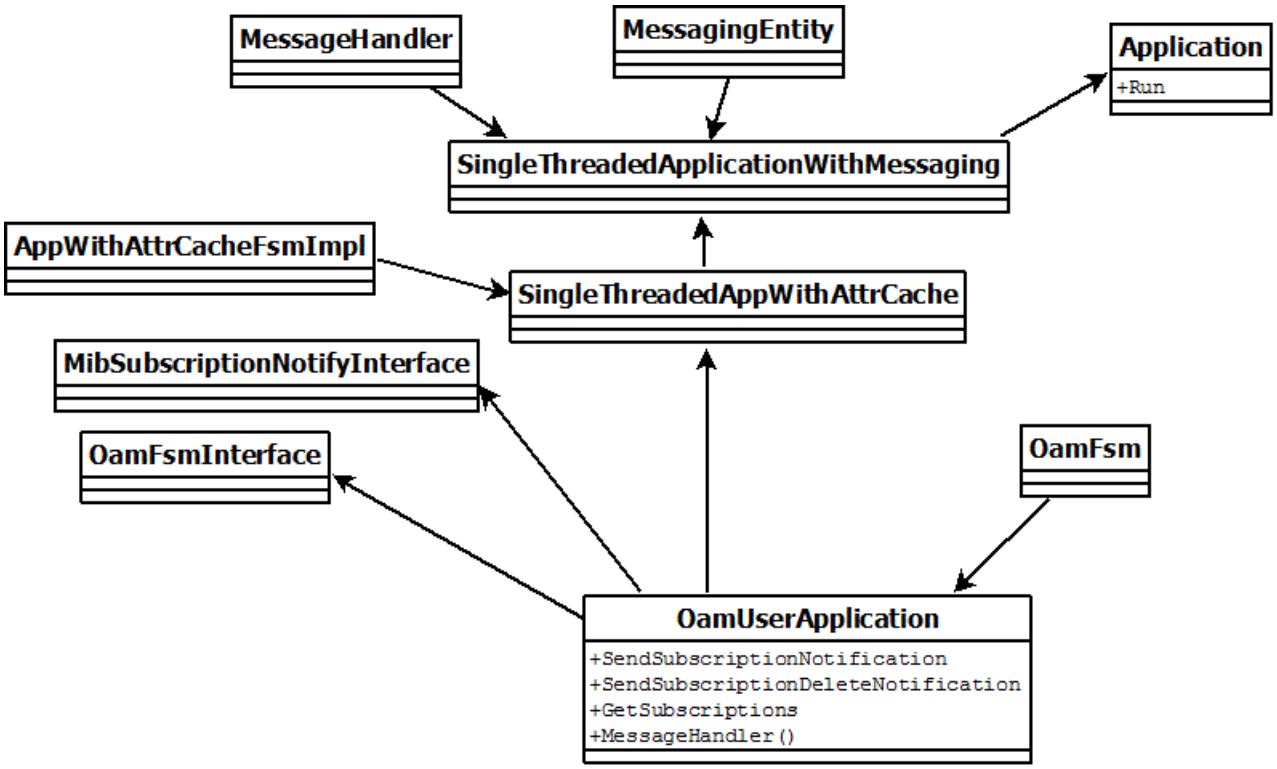

Figure 8: OAM Class Diagram

Class	Purpose
OamUserApplication	The main class that controls the functionality of OAM module.
OamFsm	Defines the FSM state transition of OAM.
OamFsmInterface	The interface class that is generated from the OamFsm which defines the actions and entry/exit points of the FSM.
MibSubscriptionNotifyInterface	Interface for informing MIB that it must send a subscription to some module which has subscribed for a particular parameter.
SingleThreadedApplicationWithMessaging	Class from which to inherit for a single threaded application.
SingleThreadedAppWithAttrCache	Extends SingleThreadedApplicationWithMessaging class to add MIB attribute retrieval/caching.
AppWithAttrCacheFsmImpl	Implementation of FSM controlling startup of an app using SingleThreadedAppWithAttrCache base class.
MessageHandler	Allows a "message handler" class to be registered within a Message Handling chain and each handler in the chain passed received messages sequentially until handled.

2.8.2. Managed Information Base (MIB)

The “MIB" is the in-RAM and non-volatile store of attributes. Following are the characteristics of MIB.

- XML is used to generate code (static tables of information)
- Run-time code uses generated code during MIB operations
  - To validate requested operations including attribute values
  - To determine whether to store the value in NV
- Attributes are defined in terms of:
  - Type(datatype)
  - Range
  - Volatility(Non-Volatile and Volatile)
- Attribute definitions are independent of objects

Managed Objects:-

- MIB currently has objects:
  - FAP.0
    - “operational” attributes for WCDMA node
  - FAP.0.FACTORY.0
    - factory set for WCDMA node
  - FAP.0.COMMISSIONING.0
    - only written at commissioning stage for WCDMA node
  - FAP.0.FAP_LTE.0
    - “operational” attributes for LTE node
  - FAP.0.FACTORY_LTE.0
    - factory set only, for example, serial number
  - FAP.0.COMMISSIONING_LTE.0
    - only written at commissioning stage
- MIB is implemented in *mib-core* library
- MIB is linked into OAM application
  - Could be in any application although OAM makes direct calls
- *mib-core* contains functionality to store values in NV (flash file)

- *mib-core* has functional API, generally not exposed
- Applications interact with MIB through messaging API (Post Office)
    - Application base classes provide helper functions

2.8.3. TR069 Client Application

Tr069 Client is used to communicate with the ACS/HeMS with the help of TR069 protocol.

The HeMS query the values of the parameters or set the values for parameters via Remote Procedure Call (RPC) methods specified as per the TR069 Broadband Forum spec.

TR069 Client is responsible for:-

- Triggering RPC request(INFORM) and responses
- Generates responses to queries from HeMS
- Validation of the parameters received from the HeMS

The protocol structure of TR069 is as follows:

CPE/ACS Management Application

RPC Methods - Remote Procedure Calls via SOAP

SOAP - XML based messaging

HTTP

SSL/TLS

TCP/IP

2.8.3.1. Class diagram for TR069 Client

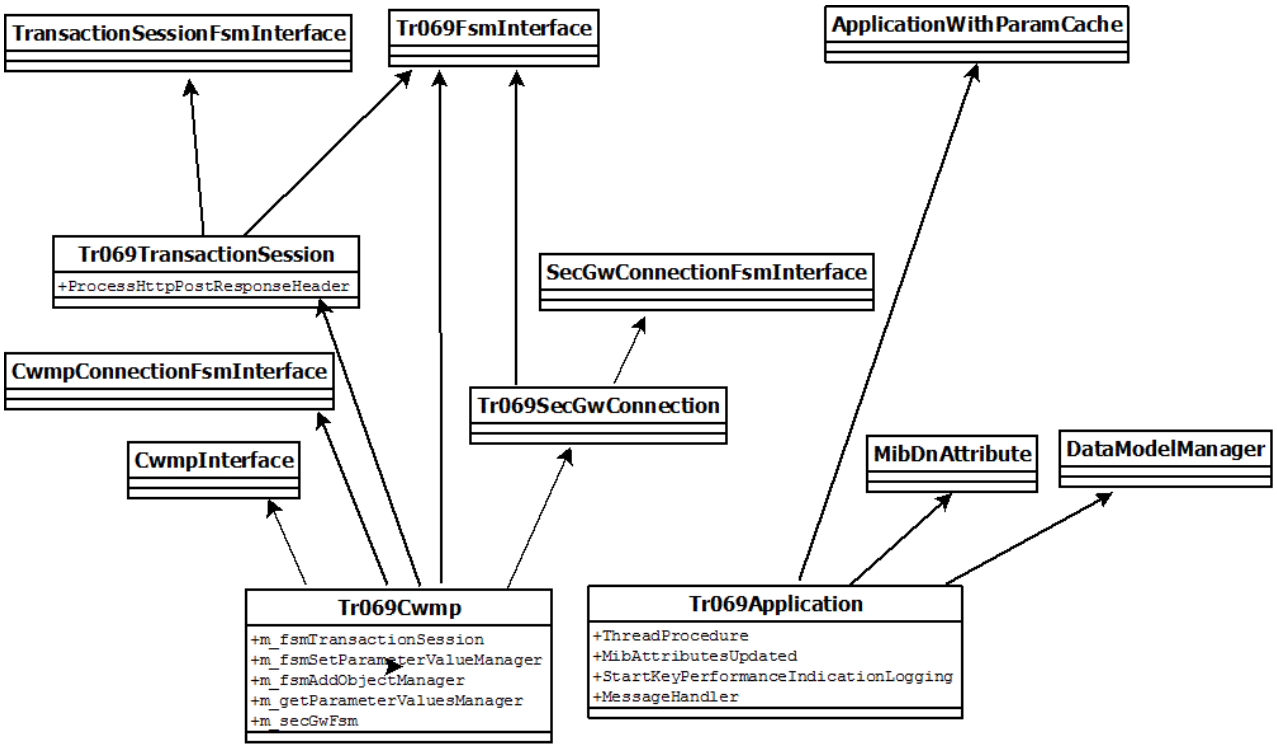

Figure 9 TR069 Client Class Diagram

Class	Purpose
Tr069Application	The main class that controls the functionality of Tr069 module
ApplicationWithParamCache	Class to add parameter retrieval/caching
MibDnAttribute	Class used to traverse through the MIB attributes
DataModelManager	Class used to verify the parameter received from HeMS against the data model
Tr069Cwmp	Class to handle the Tr069 protocol specific messages
CwmpInterface	Interface to handle the HttpResponse
CwmpConnectionFsmInterface	Interface to handle the entry/exist and actions of the FSMs.
Tr069TransactionSession	Class to handle the different RPC(Remote Procedural Calls) methods
Tr069SecGwConnection	Class to handle the Security gateway connections
Tr069FsmInterface	The interface class for TR069 FSM class.

2.8.3.2. Example call flow between HeNB and HeMS is as follows

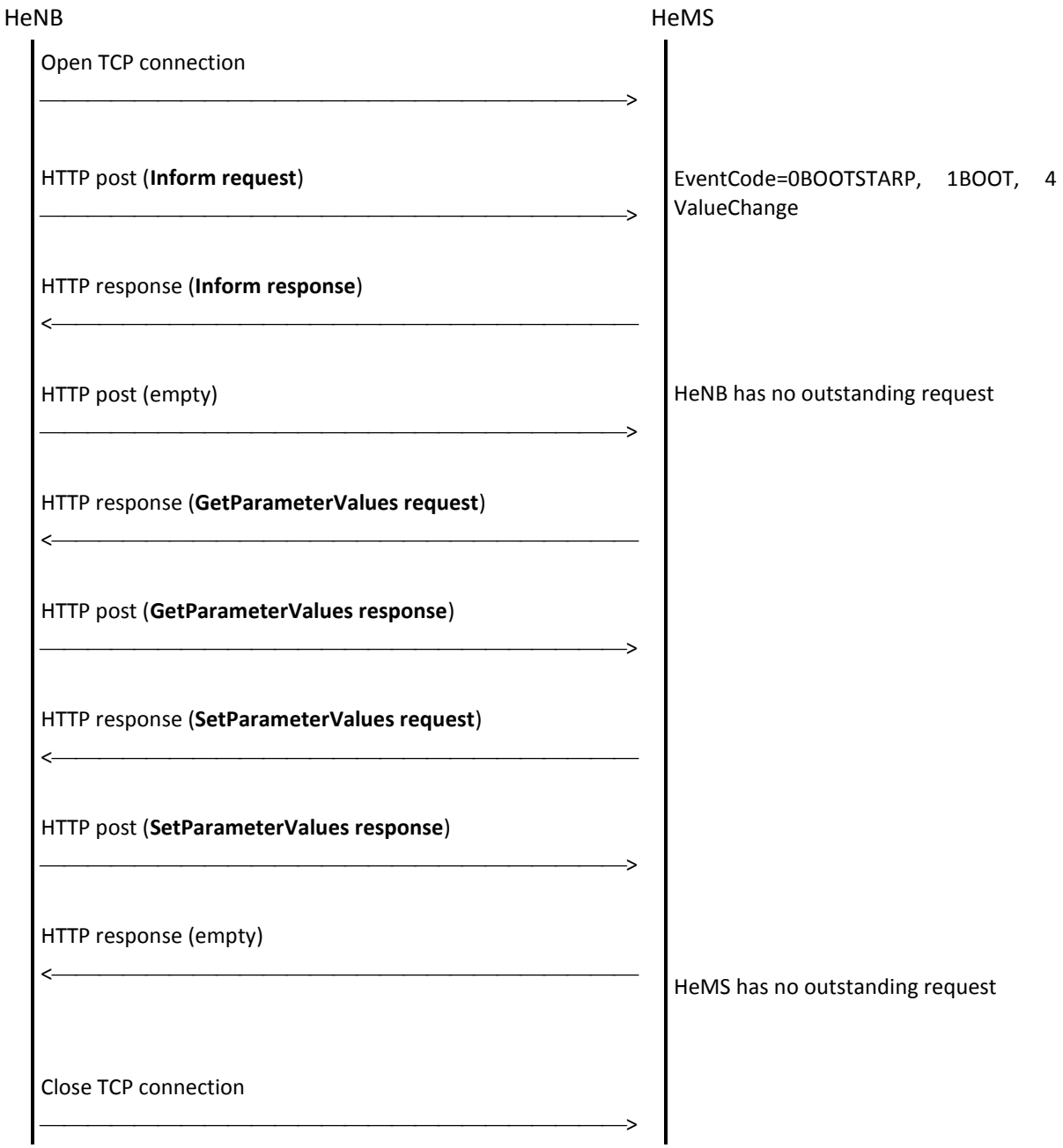

Figure 10 Call Flow between HeNB and HeMS

2.8.4. OAM-Messenger

OAM-Messenger is the Interface between OAM and Stack Manager.

During the bring up stage all the entities subscribe to the MIB parameters in groups. This is done as shown following image.

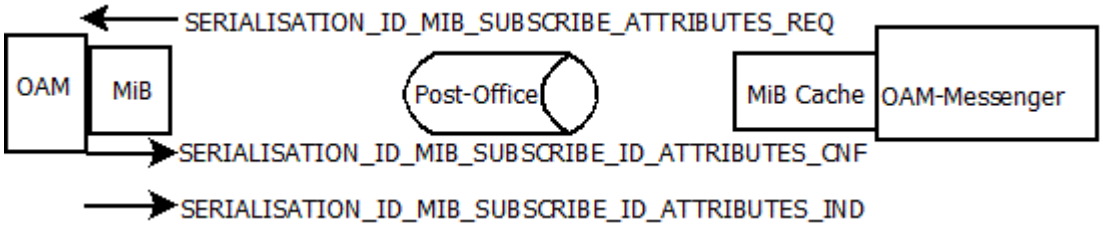

Figure 11: OAM-Messenger

The MIB framework takes care of all the data that is exchanged to attribute cache. Once the data is updated in the cache, there is an indication MIB_SUBSCRIBE_ATTRIBUTES_IND which triggers a necessary handle function. This function is as per the group (which is nothing but Subscription ID). This function sets the value in the required structures using a GetMibCache().GetMibAttribute(Param_name, variable_name).

Since this writing of values is done as part of the Handle functions, necessary structures are written as part of the interface. When a handle function is called, it writes for the mentiond interface C structures as well.

By the time HeNB is spawned, it waits for ADMIN_STATE attribute to be set to unlock to start its transactions, since the Messenger is subscribed for this parameter as part of MIB. Until the LTE_FAP_ADMIN_STATE parameter is updated in the MibAttributeCache, no subscriptions are handled. When the LTE_FAP_ADMIN_STATE case is triggered, it calls all the “handle” functions which then start mapping the data to the respective “C “structures as mentioned in the interface file. Once the mapping is complete a function call is made to SMM (eNodeB Application).

MiBCache (or MiBAttrCache) is a subset of the parameters of the MiB which are populated when they are subscribed for it by any entity.

The subscription IDs used are:

- LTE_SUBSCRIPTION_INITIAL_CFG
- LTE_SUBSCRIPTION_ADMIN_STATE

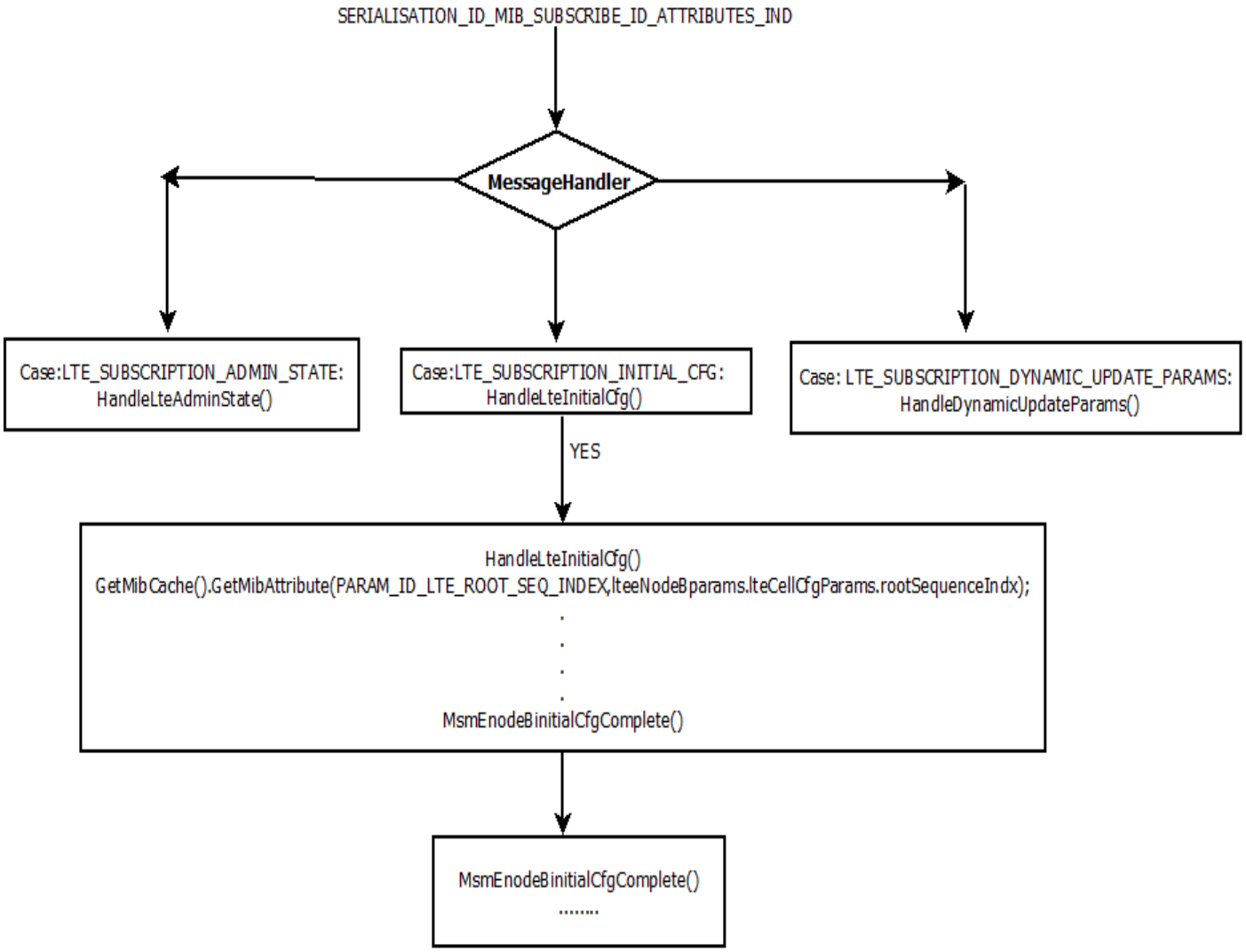

Figure 12: Parameter handling in OAM-Messenger

2.8.4.1. Sequence diagram for Get Parameter Value and Set Parameter Value

Overall flow of a SetParamValue(SPV) from HeMS is as described as follows:

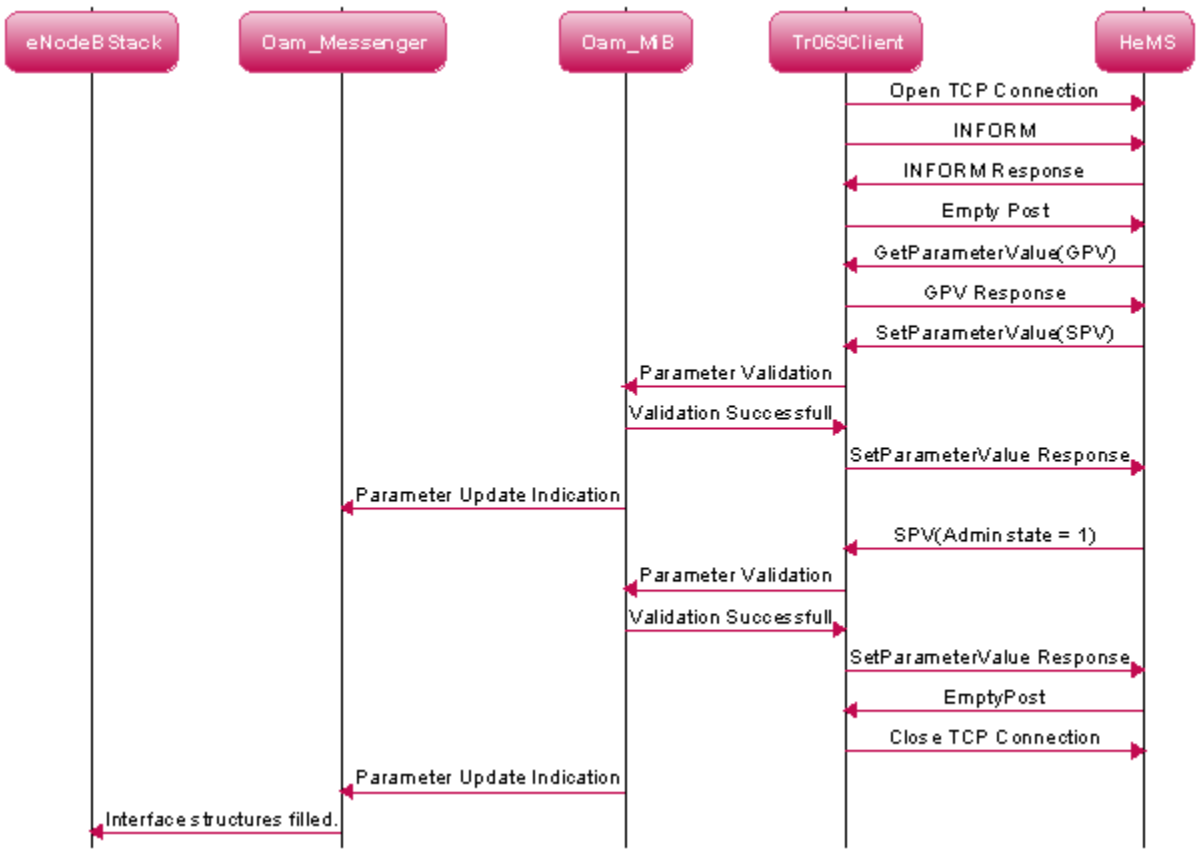

Figure 13: Sequence Diagram for Set Parameter

2.9. Interfaces

Table 9: Interfaces

Name	Parameters	Purpose
MsmEnodeBinitialCfgComplete	NA	Stack Manager call this to indicate OAM-Messenger about the completion of initial configuration.
MsmAdminStateChanged	NA	Inform Stack manager if admin state is changed.
GetMibAttribute	PARAM_ID_XXXX, variable	Get the MIB Attribute
SetMibAttributes	Dn, value, toentity	Set the MIB attribute

2.10. Performance Management (Monitoring)

PM is comprised of gathering network statistics, evaluating system performance under both normal and degraded conditions and altering system mode of operation. PM entity supports for data collection. This support is based on direct polling or indirect polling. Performance of the device is determined by examining performance counters.

2.11. Functional Description

Key Performance Indicators (KPI) is set of metrics that allow a reliable and complete assessment of network performance in the evaluated scenario. It has an important role in performance management, collection of statistics, metrics. Key performance indicators allow a conclusive comparison of a set solution. We collect them on the FAP and report them (when asked) to the HMS over TR069 or optionally it is retrieved through CLI. KPIs have a collection period for example, 24 hours, and when the KPIs are requested by the TR069 server, the FAP responds with the KPIs from the previous 24hrs (midnight-to-midnight).

The KPI architecture allows for multiple managing entities with differing KPI collection requirements. For example, the TR069 and Service API entities configure the KPI mechanisms to collect different KPIs at different intervals.

To achieve this each managing entity configures OAM with the list of KPIs it is interested in and the period over which they must be collected.

As each period expires, OAM queries the entities that collect the KPIs and instructs them to send their results to the managing entity.

The entities that perform the work of collecting the KPIs do so until they are told to report them to some managing entity. Each time the values are reported they are reset to zero (for that particular managing entity).

2.12. KPI Configuration

A managing entity uses KpiConfigureReq to tell OAM which KPI groups it is interested in and the period over which to collect those KPIs.

OAM runs a repeating timer (particular to that managing entity) which is used to request KPI values from the collecting entities by sending them a KpiReadReq. This contains the managing entity to which the collector must respond.

2.13. KPI Collection

The entities that actually update the KPI values do so continually and have no real knowledge of what those values are used for or who cares about them.

Each entity that collects KPIs registers those KPIs with OAM at startup so that OAM knows which entity to query to get the current values.

When a KpiReadReq is received the entity sends its current values to the specified managing entity and resets its values to zero.

2.14. KPI Storage

The period over which KPIs are read (as configured by the managing entity) is not necessarily (or even normally) the same as the period over which the external manager for example, the Femto Gateway, requires the results. The external manager may request KPIs on an irregular interval or on an interval that is longer than the period over which accept a loss of KPI data due to power outage.

For this reason, KPIs are periodically stored in FLASH. As the period for doing this is likely to be dependent on the managing entity it is left to the managing entity to do this through use of the KpiLogFileMgr class.

The number of measurements stored must be sufficient to allow the FAP to retrieve all KPIs for the previous (external) reporting period.

2.15. Design Details

As part of Common Platform, classifications/organizing of counters are maintained at OAM.

Counters are classified into two types based on the performance impact of stack when PM is enabled:

- *NonL2PMCounters (UpperArm Counters):*
  For example: RRC, eGTP, SCTP, TUCL counters

- *L2PMCounters(LowerArm Counters):*
  For example: RLC, MAC counters

There are couples of vendor specific configuration parameters defined to enable/disable the L2PMCounters.

- *CollectionMethodForPM: All (L2+L3)/L3Only*
- *L2PMCollectionEnable: True/False*

Reporting Mechanism of NonL2PMCounters:
These counters are reported to OAM by means of a function call, immediately when KPI noted for change.

2.15.1. Class Diagram

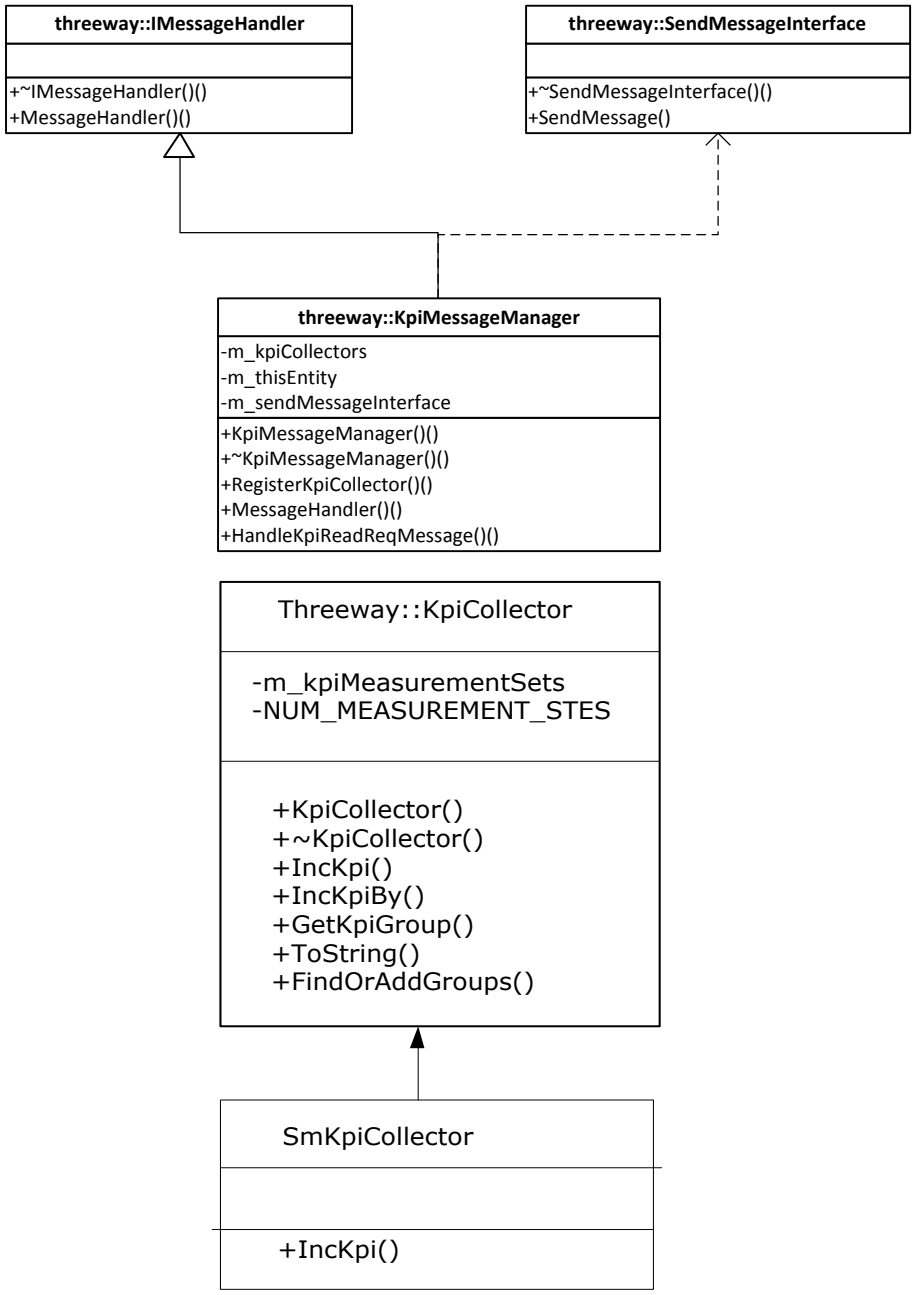

Figure 14: Class Diagram

CLASS	PURPOSE
KpiGroup	Aggregates a set of KPIs and provides increment and get methods. If serialisable, it can be reported to a management entity (in a KpiReadCnf) and written to flash
KpiCollector	Class that has a container of KpiGroup objects that may be updated by code that actually updates KPIs.

2.15.2. Structures

Following terminologies are used for KPI.

KPI ID:

Performance Management counter is called Key Performance Identifier (KPI) .In KPI Module, each KPI is identified by a unique ID called KPI ID. KPI ID is an enum value. KPI IDs are defined in KpiTypes.h file.

For example, KPI_ID_UNSUCCESSFUL_RRC_CONN_REEST, KPI_ID_SUCCESSFUL_RRC_CONN_REEST, ...

Naming Convention: KPI_ID_XXXX, where XXXX is the Procedure Name.

KPI Group ID:

In KPI Module, KPI Group is a set of KPI IDs. KPI Group is identified by a unique id called KPI Group ID. KPI Group ID is an enum value. KPI Group IDs are defined in KpiTypes.h file.

For example, KPI_GROUP_ID_ATTEMPTED_RRC_CONN_REEST,

KPI_GROUP_SUCCESSFUL_RRC_CON_REEST, …

Naming Convention: KPI_GROUP_ID_XXXX, where XXXX is the Procedure Name.

KPI Group ID definition consists of prefix KPI_GROUP_ID and procedure name.

In KPI Module, KpiGroup class is defined in KpiGroup.h and KpiGroup.cpp files contains actual counts for the KPI IDs within the group and methods for updating and accessing the KPI IDs information stored in a KPI group.

KpiGroupDefinition:

This structure contains the information of KPI Group like ID of KPI Group and description of KPI group.

This structure is defined in file KpiTypes.h.

For example, {KPI_GROUP_ID_ATTEMPTED_RRC_CONN_REEST, “AttemptedRrcConnReest”}

KpiDefintion:

This structure contains the information of KPI ID like ID of KPI, KPI Group to which KPI ID belongs, type of KPI ID and description.

For example, {KPI_ID_ATTEMPTED_RRC_CONN_REEST, KPI_GROUP_ID_ATTMEPTED_RRC_CONN_REEST, KPI_TYPE_U32_COUNT,”ALL”}

KpiCollector:

A set of KPI Groups associated with one entity (OAM, SM, REM), called KPI Collector. In KPI Module, KpiCollector class is defined in files KpiCollector.cpp and KpiCollector.h.

The KpiCollector class contains KPI Groups and a set of methods for accessing the KPI Groups.

Following structures are used.

Each KPI has a unique ID and KPIs are organized into KPI groups.

Each KPI is of two types either it by count or by increment, structure defined as,

```
typedef enum
{
   KPI_TYPE_U32_COUNT = 0,
   KPI_TYPE_U32_INC_BY_N
} KpiType;
```

KPI structure is defined as,

```
typedef struct
{
   KpiId      kpiId;
   KpiGroupId kpiGroupId;
   KpiType    kpiType;
   const char * description;
} KpiDefinition;
```

Each KPI is put under a KPI group, whose structure is defined as:

```
typedef struct
{
   KpiGroupId kpiGroupId;
   const char * description;
} KpiGroupDefinition;
```

typedef enum

{

KPI_ID_LTE_RRC_ATTCONNESTAB_SUM

KPI_ID_LTE_RRC_ATTCONNESTAB_EMERGENCY

KPI_ID_LTE_RRC_ATTCONNESTAB_HI_PRIO_ACCESS

KPI_ID_LTE_RRC_ATTCONNESTAB_MT_ACCESS

KPI_ID_LTE_RRC_ATTCONNESTAB_MO_SIGNAL

KPI_ID_LTE_RRC_ATTCONNESTAB_MO_DATA

KPI_ID_LTE_RRC_ATTCONNESTAB_DE_TO_ACCESS

KPI_ID_LTE_SUCCCONNESTAB_SUM

……..

…..

}

2.15.2.1. Sequence Diagram

This section describes the following sequence diagrams related to Performance Management.

2.15.2.1.1. KPI Application Startup

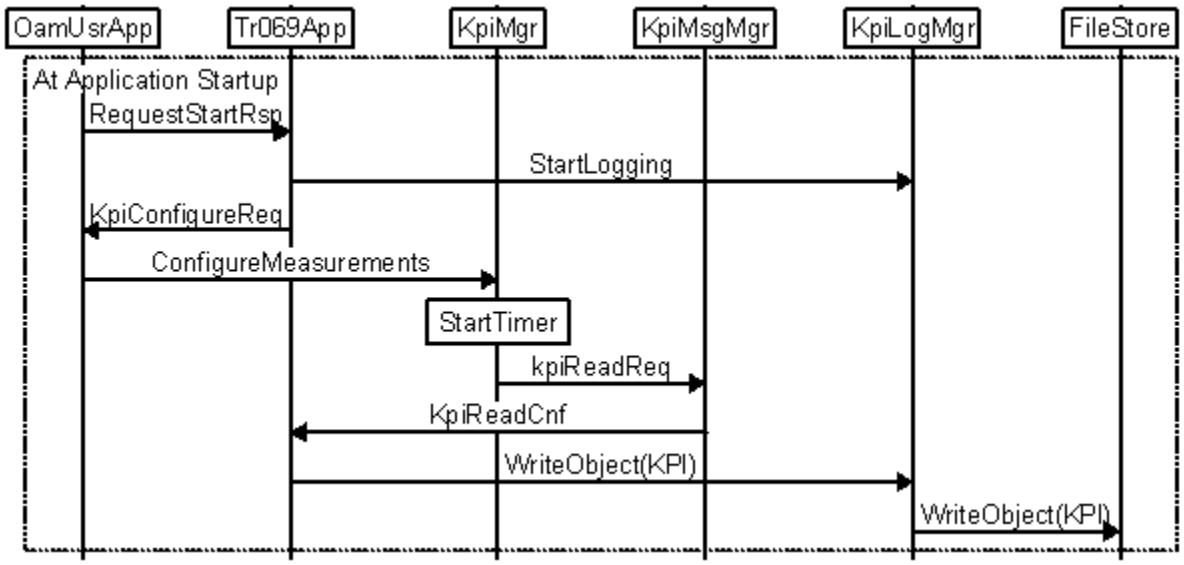

Figure 15: KPI Application at Startup

1) At Application startup, OAMUserApp starts the TR069App with RequestStartRsp.
2) TR069 App sends an indication StartsLogging to KpiLogMgr.
3) TR069 App sends KpiConfigureReq to OAMUserApp for configuring KPI groups.
4) OAMUserApp sends request to KPIMgr for collecting the KPI group's info at periodic interval.
5) KPI Manager starts the Timer based on periodic intervals and whenever the timer expires, it requests KpiMsgMgr for getting KPI info.
6) KPIMsgMgr collates the KPI info and sends it to TR069 in KpiReadCnf object.
7) TR069 collects the KPI's info and writes into InfiniteFileStore.

2.15.2.1.2. KPI Updating

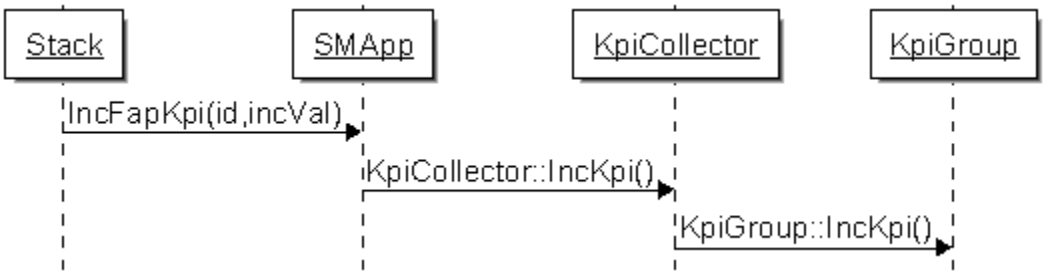

Figure 16: KPI Updating

1) Stack informs SMApp through IncFapKpi when any event KPI is noted at the stack.
2) SMApp in turn forwards KPI info to KPICollector.
3) KPICollector segregates the KPI into KPI group and updates the KPI counter.

2.15.2.1.3. Retrieval of KPI

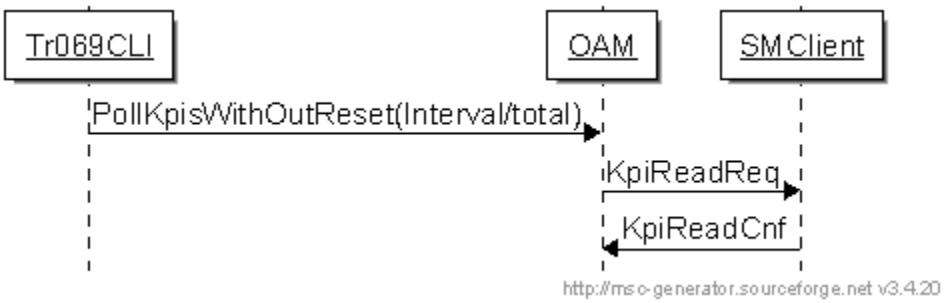

Figure 17: Retrieval of KPI

1) CLI requests for the KPI info through PollKpiWithoutReset for pre-defined interval.
2) OAM sends SMClinet for KPI info through KpiReadReq.
3) OAM collates all the KPI info received in KpiReadCnf and dumps the data on to terminal.

2.15.2.1.4. PM File Retrieval

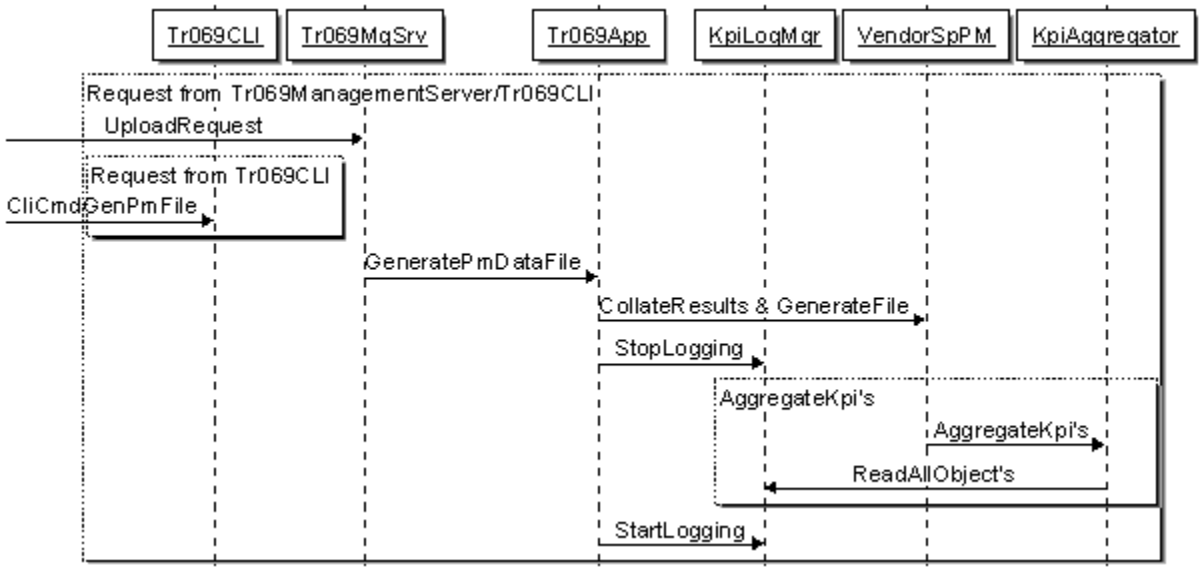

Figure 18: PM File Retrieval

1) TR069 sends upload request for KPI to the Tr069MgServer.
2) Tr069MgServer sends GeneratePMDataFile request to Tr069App.
3) Tr069App request KpiLogMgr to stop logging.
4) KpiAggregator collates all KPIs from VendorSpPM and dumps the objects into log.
5) TR069 collects the logs from VendorSpPM.

2.16. Interfaces

Stack Application (SMApplication) make a direct function call to the OAM Application to notify the KPI change.

Interface Name	Parameters	Description
IncFapKpi	KPI ID(For example, RRC.establishment.Sum), incVal (increment step in +/-)	To notify the KPI change to the OAM.

With this API, it is ensured that counter's grouping/organizing are pushed to be customer-specific. Stack Application notify counter changes and it is up to OAM to organize/group the counter data.

2.17. Files Added/Modified

Applications/Modules Affected:

1. OAM
    - Manages the configuration and collection of all KPIs.
2. TR069
    - Configures OAM for the KPIs it maintains and reports them to the Framework.
    - Manages the storage of KPIs in FLASH.
3. All applications that have the knowledge to populate specific KPIs.
    - Each registers their KPIs with OAM and continually collects KPIs.

Files Modified:

libs/platform/	KpiTypes.h
libs/system/common/	KpiGroup
	KpiCollector
	KpiAggregator
	KpiLogFileMgr
libs/messaging/messages/common/	KpiConfigureReq
	KpiRegisterReq
	KpiReadReq
	KpiReadCnf
apps/fap/management/oam/	KpiManager
apps/fap/management/tr069-v2/	NsnPerformanceMeasurement
enbapp/src	wr_kpi.h
	wr_kpi.c – This contains the methods related to counters.

2.18. Configuration

For test purposes, KPI_REPORT_DURATION is set to something lower than the value specified by PMGranularityPeriod. However note that when a TR069 configuration plan is downloaded, this value is overwritten if KPI_REPORT_DURATION is present in the configuration plan.

For example, set report duration to 10 minutes:

```
/mnt/dbx/cli -c “mib.set  KPI_REPORT_DURATION 10”
```

2.19. Results

The current KPI values are requested through the CLI command:

oam.pollkpis <interval|total>

'interval' gives the KPIs for the current collection period, since they were reset at the end of the last collection period.

'total' gives the total value since power on.

This triggers OAM to poll for all current KPI values without resetting them. Check the OAM trace file for the results.

2.20. Stack Manager (SM)

This module is responsible for configuring and managing all the protocol layers in the eNodeB Protocol Stack. This module's responsibilities are summarized as follows:

- Configure all the Trillium layers in eNodeB stack.
- Configure relevant upper and lower SAPs between individual layers.
- Trigger binding procedure across all layers.
- Enable debugs and alarms functionality.
- Trigger S1AP Setup Connection Procedure.
- Trigger Cell Setup Procedure.
- Triggers dynamic configuration depending on the OAM configuration update.

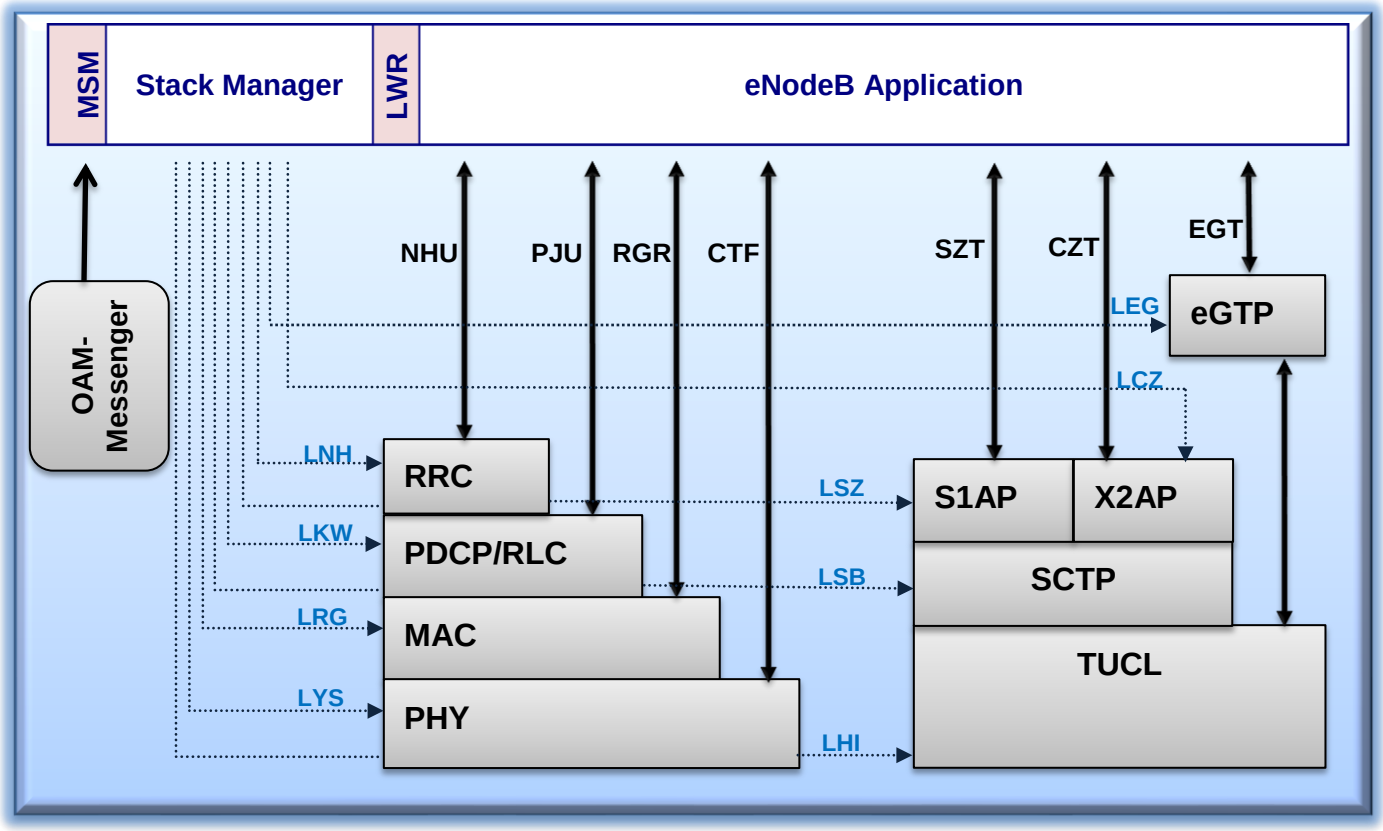

Figure 19: LTE eNodeB Application and Stack Manager

2.21. Functional Description

The Stack Manager configures individual layers and their respective upper and lower SAPs through respective Layer Manager (LM) Interfaces. For example, Stack Manager interacts with RRC over LNH Interface; similarly, each layer exchanges primitives over its respective layer manager interface. Above mentioned figure shows the LM interfaces (LXX) of each layer (where, XX is the unique prefix of each layer) which interacts with Stack Manager.

Stack Manager fetches values for the configuration related parameters from a flat file (wr_cfg.txt) and wr_smallcell.c. This includes, cell configuration, debug flags, and neighbor cell configuration and handover timer values. Neighbor cell configuration triggers the X2AP setup towards the peer eNodeB.

Apart from configuring, LM is used by the Stack Manager to enable debug information and alarms for each layer and trigger Bind procedure. After all the layers in the stack are configured and bound; SM triggers the Control Request primitive towards eNodeB Application to establish S1AP Connection Establishment procedure. On the positive confirmation that S1AP Connection is setup successfully, SM triggers Control Request toward eNodeB Application to initiate Cell Setup.

2.22. Design Details

The Stack manager has two modules.

1. Stack Manger part which manages and controls the Stack including the eNodeB application call Stack Manager Module (SMM), which has the interface with all the layers.

2. Stack Manager which receives configuration details and updates from OAM and passes to SMM is called Messenger (MSM).

The interface between the SMM and Messenger is called MSM (Messenger to Stack Manager). SMM and Messenger are tightly coupled through MSM interface.

Stack Manager has the state machine and perform all its tasks sequentially.

Following primitives are executed in sequence of Stack Manager's state machine. When Stack manager completes the state machine, it moves to ideal state and waits for inputs from OAM also and indications from the stack.

Functions	Description
MsmEnodeBinitialCfgComplete	Routine to start induction for state machine.
smWrProcSm	Stack Manager State Machine Start
wrSmHdlTuclCfgEvent	Routine to configure TUCL Layer
wrSmHdlSctpCfgEvent	Routine to configure SCTP Layer
wrSmHdlS1apCfgEvent	Routine to configure S1AP Layer
wrSmHdlEgtpCfgEvent	Routine to configure eGTP Layer
wrSmHdlAppCfgEvent	Routine to configure eNodeB Application Layer
wrSmHdlRrcCfgEvent	Routine to configure RRC Layer
wrSmHdlPdcpCfgEvent	Routine to configure PDCP Layer
wrSmHdlRlcCfgEvent	Routine to configure RLC Layer
wrSmHdlMacCfgEvent	Routine to configure MAC Layer
wrSmHdlCLCfgEvent	Routine to configure Convergence Layer
wrSmHdlEnbDebugs	Routine to enable debug prints
wrSmHdlEnbAlarms	Routine to enable alarms
wrSmHdlBndS1apStack	Routine to trigger Bind procedures in the S1AP stack
wrSmHdlBndEgtpStack	Routine to trigger Bind procedures in the eGTP stack
wrSmHdlBndRrcStack	Routine to trigger Bind procedures in the RRC stack
wrSmHdlInitS1Setup	Routine to trigger S1 Setup Procedure
wrSmHdlInitCellCfg	Routine to trigger Cell Setup Procedure
MsmConfigComplete	Routine to inform configuration is completed

Note: The Brown color items are newly added as part of common platform.

2.23. Sequence Diagram

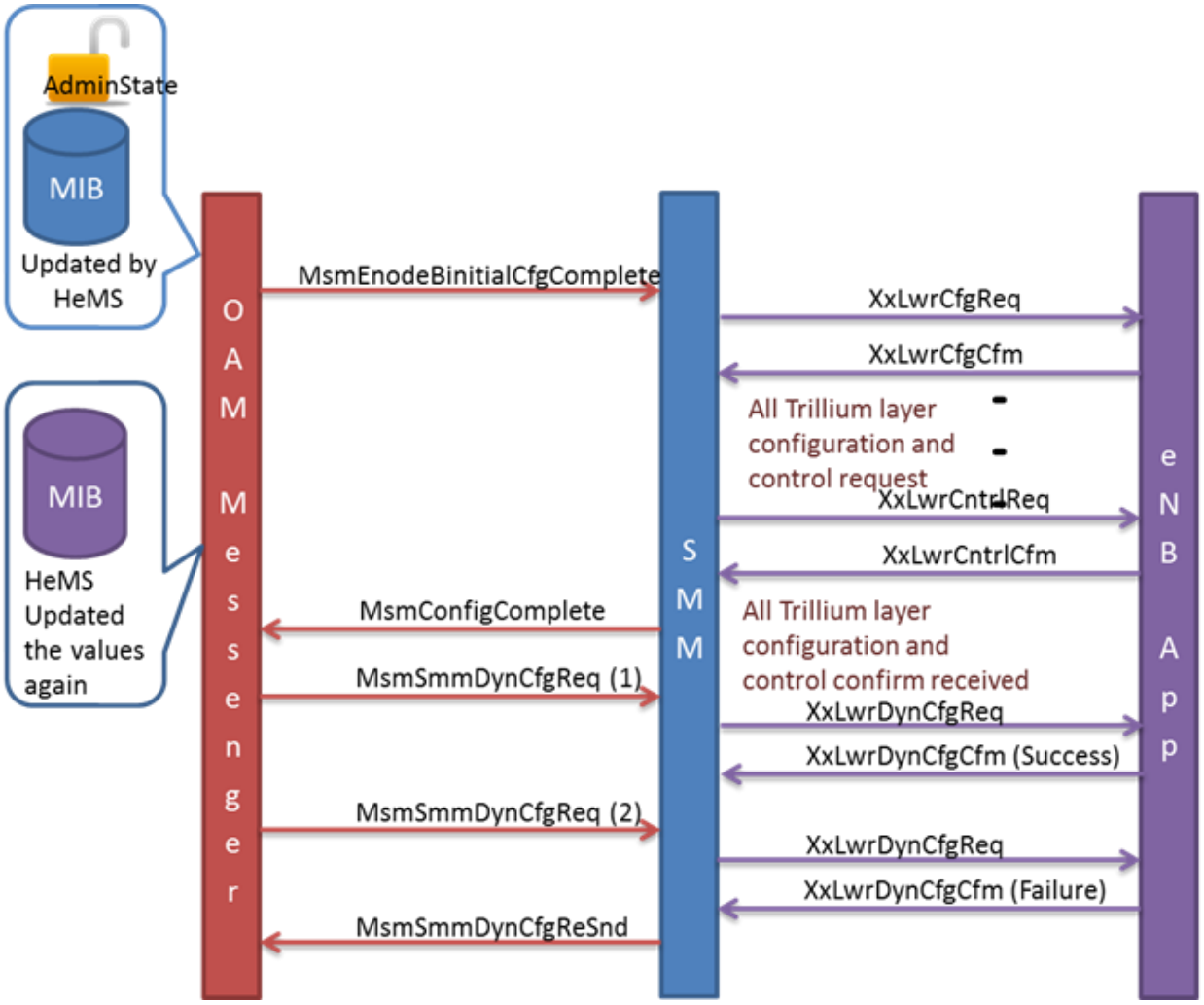

Figure 20: Configuration Sequence Diagram

2.24. Files Added/Modified

Additions

enbapp/src/msm_common.h – This has interface structure for MSM interface.

enbapp/src/wr_smm_cmnplt_enbapp.c – This has configuration of WR and parameters are read from OAM-Messenger.

Modifications

enbapp/src/wr_ex_ms.c

enbapp/src/wr_lmm.c

enbapp/src/wr_lwr.c

enbapp/src/wr_lwr.h

enbapp/src/wr_ptmi.c

enbapp/src/wr_smm_enbapp.c

enbapp/src/wr_smm_init.c

enbapp/src/wr_smm_init_merged.c

enbapp/src/wr_smm_init.h

enbapp/src/wr.mak

2.25. Compile Options

Newly added files and common platform shared objects are added in make file.

2.26. Interfaces

The interface between the Stack Manager and all Trillium layers are loosely coupled (Message based communication through SSI) except eNodeB application. eNodeB Application and Stack Manger are light weight loosely coupled (Buffer pointers, parameters are packed and sent through SSI).

1. SMM module has interface (LXX) with all Trillium layers.

   Xx is Trillium layer prefix.

Name	Description	Parameters
XxMiLxxCfgReq	Configuration request.	Pst *pst, XxMgmt *cfg
XxMiLxxCfgCfm	Configurations confirm.	Pst *pst, XxMgmt *cfm
XxMiLxxStsReq	Statistics request.	Pst *pst, Action action; XxMgmt *sts
XxMiLxxStsCfm	Statistics confirm.	Pst *pst, XxMgmt *cfm
XxMiLxxStaReq	Status request.	Pst *pst, XxMgmt *sts
XxMiLxxStaCfm	Status confirms.	Pst *pst, XxMgmt *cfm

Name	Description	Parameters
XxMiLxxStaInd	Status indication.	Pst *pst, XxMgmt *sta
XxMiLxxCntrlReq	Control request.	Pst *pst, XxMgmt *cntrl
XxMiLxxCntrlCfm	Control confirms.	Pst *pst, XxMgmt *cntrlCfm

2. SMM has separate primitive to configure dynamic updates received from OAM-Messenger.

Name	Description	Parameters
XxLwrDynCfgReq	Dynamic configuration request	Pst *pst, Header *hdr, Void *dynCfg, U32 dyCfgtype, Bool cfgPriority
XxLwrDynCfgCfm	Dynamic configuration confirmation	Pst *pst, Header *dynCfmHdr, CmStatus *dynCfgCfm

3. Messenger has two interfaces, one towards OAM and other towards SMM (MSM interface). Following are the MSM interface primitives. These primitives control the state machine of SM.

Name	Description
MsmConfigComplete	Routine to inform configuration is completed
MsmAdminStateChanged	Routine to inform Admission state is changed
MsmEnodeBinitialCfgComplete	Routine to start induction for state machine.
MsmDynamicConfiguration	Routine to send the Dynamic MIB updates to stack as reconfiguration.

3. Appendix A (Design Decisions)

3.1. Interface between OAM-Messenger and Stack Manager

For the integration of common platform into TotaleNodeB, there is a need for the communication between the OAM-Messenger and Stack Manager to propagate the ***static configuration*** parameters into the stack. For this purpose the following approach is taken:

1. The OAM-Messenger maintains a global structure, which gets updated from the MIB.
2. Once the MIB updating is complete and the "AdminState" is set to "unlocked" by HeMS, the OAM-Messenger indicates the Stack Manager to start configuring all the layers. After configuration completes, Stack Manger indicates (Configuration Complete) to OAM-Messenger. This interface is named as MSM (Messenger to Stack Manager).
3. For configuration of different layers Stack Manager uses the same global structure maintained in the OAM-Messenger.
4. A separate file is maintained for the handling of configuration parameter received from OAM (*wr_smm_cmnplt_enbapp.c*).

For ***Dynamic Configuration*** following approach has taken:

1. Dynamic updates are accepted if the "AdminState" is unlocked and configuration of different layer is completed. OAM-Messenger does not trigger dynamic updates until initial configuration is done.
2. If there are any updates before initial configuration, OAM-Messenger pushes the updates into an array. After receiving configuration complete indication (success/failure) from Stack Manager, OAM-Messenger pushes push only the structure info (say subscription Id) along with enum and deferred/immediate parameters.
3. When admin state is set to locked, the OAM-Messenger performs a function call to Stack Manager and Stack Manager sends a config message to the eNodeB application to perform Cell Reset.
4. For dynamic updates separate primitive has introduced, which takes the updated structures as Void pointer.

3.2. Interface between Stack Manager and eNB App

Static Update

The LwrMgnt structure is used for communication between Stack Manager and eNB App. Since the structure is a huge one, the Lightweight loosely coupled mechanism is used as an interface between Stack Manager and eNB App.

Dynamic Update

For dynamic update the modified child structure of LWR is passed as “void pointer” to the eNB App.

3.3. KPI Interface

SM application defines a macro to update the KPIs. eNB Application, stack layers make use of this macro to make a direct function call to the SM application to notify the KPI change.

SM application in turn updates the KPICollector using KPICollector interfaces. Stack Application notify counter change and it is up to OAM to organize/group the counter data.

Interface Name	Parameters	Description
IncFapKpi (Macro)	KpI ID (For example, RRC.establishment.Sum) incVal (increment step in +/-)	To notify the KPI change to the OAM

This approach ensures that there is a generic interface to update counters, how the counters are grouped/organized, and are pushed to customer specific implementations. This requires minimal change to the Macro and defining the KPI-grouping.

Impact on Customer with their OAM

With common platform counter implementation, the customer who wants to integrate their own OAM needs to do the following:

1) There is a static one-one mapping between the Radisys counter and customer specific counter.
2) Counter group name derivation is again customer specific.

www.radisys.com