Software Design and Development Specifications

Product / Module: SPI Probe

Version: DRAFT

Date: 2nd Feb 2016

History/Change Log:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Prepared By | Changes/Comments | Reviewed By |
| Draft | 2nd Feb 2016 | * Ashif Jamadar * Harsh Manyase | First release for initial review |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

[1 Introduction 4](#_Toc442374553)

[2 Purpose of Document and Audience 4](#_Toc442374554)

[3 References 4](#_Toc442374555)

[4 Glossary of Terms 5](#_Toc442374556)

[5 High Level Architecture 6](#_Toc442374557)

[6 Detailed Design 7](#_Toc442374558)

[6.1 Design Process and Design Considerations 7](#_Toc442374559)

[6.2 Performance Goals and Their Impact on Design 8](#_Toc442374560)

[6.3 Assumptions and Dependencies 8](#_Toc442374561)

[6.3.1 Assumptions 8](#_Toc442374562)

[6.3.2 Dependencies 8](#_Toc442374563)

[6.4 Call Flows/Time Diagrams/Application & Process States 12](#_Toc442374564)

[6.5 Application/Module Interfaces and I/O Definitions 16](#_Toc442374565)

[6.5.1 Interfaces 16](#_Toc442374566)

[6.5.2 Inputs 16](#_Toc442374567)

[6.5.3 Outputs 17](#_Toc442374568)

[6.6 Internals 18](#_Toc442374569)

[6.6.1 Multi-Threading 18](#_Toc442374570)

[6.6.2 Buffers/Queue Management 19](#_Toc442374571)

[6.6.3 Application Self-Health Monitoring 19](#_Toc442374572)

[6.6.4 Database Design 20](#_Toc442374573)

[6.6.5 Others Sources 21](#_Toc442374574)

[6.7 Auxiliary/Platform Modules 21](#_Toc442374575)

[6.7.1 Configuration Management 21](#_Toc442374576)

[6.7.2 Logging 21](#_Toc442374577)

[6.7.3 Alarms and Notifications 22](#_Toc442374578)

[6.7.4 Reporting 22](#_Toc442374579)

[6.8 Hardware and Software Environment Requirements 22](#_Toc442374580)

[6.8.1 Development Environment 23](#_Toc442374581)

[6.8.2 Testing Environment 23](#_Toc442374582)

[6.8.3 Production Deployment Environment 24](#_Toc442374583)

[7 High Level Test Plan and Testing Requirements 24](#_Toc442374584)

[8 Considerations for Troubleshooting and Production Support (Sample Scenarios and work flow) 25](#_Toc442374585)

[9 To Do’s and Roadmap 26](#_Toc442374586)

# Introduction

UserApplication is something that reads the particular stream of data in the network coming from SP Probe Application then decodes that data (i.e Identify) and Write it into the File and Database as per its format. Type of message Format is discussed later in Documentation ().

# Purpose of Document and Audience

This document is to understand the flow of Application where all the scenarios are mentioned. Which will make User friendly to user who is going to use this Application. The Complexities are made easy to understand when user go through this Document.

# References

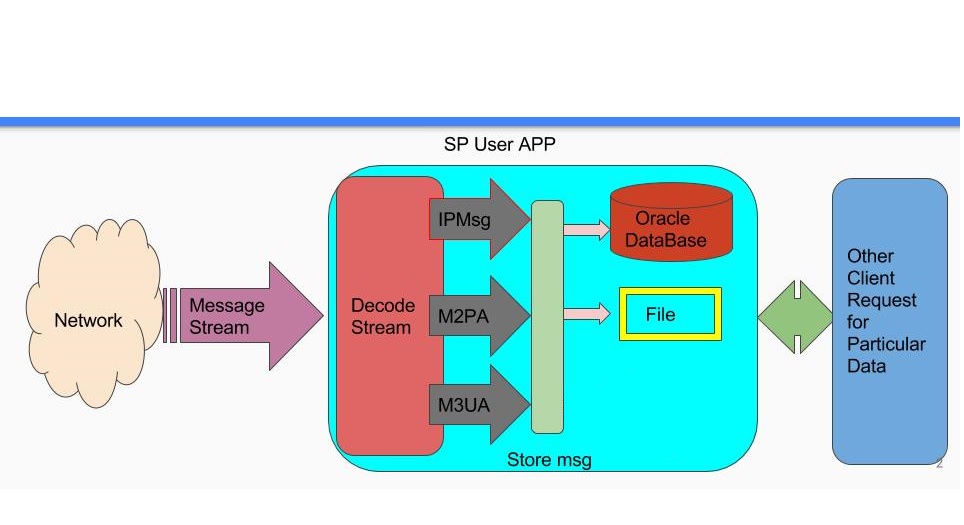
M2PA: IETF RFC 4165

M3UA: IETF RFC 4666

# Glossary of Terms

* SS7: Signaling System 7
* SCTP: Stream Control Transmission Protocol
* M2PA: MTP2 user peer-to-peer Adaptation protocol
* M3UA: MTP3 user adaptation protocol
* SCCP: Signaling Connection Control Part
* TCAP: Transaction Capabilities Application Part
* Opc: Originating Point Code
* Dpc: Destination Point Code
* Si: Service Indicator
* Ni:Network Indicator
* Mp: Message Priority
* Sls: Signaling Link Selection
* Bsn: M2PA Backward Sequence Number
* Fsn: M2PA Forward Sequence Number
* Pri: M2PA Priority
* Sio: Signaling Information Octet
* Sif: Signaling Information Field

# High Level Architecture

****

When stream is received from the network (SP Probe), the stream is read (Decode) and identify the type of message i.e. IPMsg/M2PA/M3UA stored in message queue/Buffer and then it is written (stored) in File/Database respective to the type of message.

The Message stored in File/Database is also used for communication with other Application if they required the data from respective File/Database by connecting Our Application to other Application.

# Detailed Design

## Design Process and Design Considerations

1. UserApplication gets connection With SP Probe.

* Initial request send/receive from UserApp to SP Probe

UserApp (Initial request) SP Probe.

* Connection Established between SP Probe and UserApp.

SP Probe (Ack, Port) UserApp.

* Exchange of Port and IP Address between SP probe and UserApp.

UserApp (send Port, IP Address) SP Probe.

* Connection Established between SP Probe and UserApp.

SP Probe (getport (), getIPAddress ()) UserApp.

2. Communication Using CommonUtil

* Contains Type of Message (IP/M2PA/M3PA) send by SP Probe.
* Types of message are described in SPProbe\_REQ.

3.Read Type of message From SP Probe.

* i.e.(IP/M2PA/M3UA)

4.Set Type of messages in Com.Pojos package.

* SS7\_Message\_IP.
* SS7\_Message\_M2PA
* SS7\_Message\_M3UA
* SPProbe\_Error.

5.Write to particular File or Database or Other Application.

* FileUtil/Database.

## Performance Goals and Their Impact on Design

//TO DO

## Assumptions and Dependencies

### Assumptions

The Stream(Input message) is provide from SP Probe Application to UserApplication .The Message received from them will be in the format of IP/M2PA/M3UA.They can also provide the message stream which contain different message format that is TCAP/SCCP/etc.

### Dependencies

Dependencies can be declared in two modes called “Direct Dependencies” and “Output dependencies”.

#### Direct Dependencies

The SP Probe provides the Stream where it includes the type of message.

Type of Message.

|  |
| --- |
| 1. SS7Message\_IP. |
| 2. SS7Message\_M2PA. |
| 3. SS7Message\_M3UA. |

(Structure of Message coming from SP Probe)

1. **SS7Message\_IP.**

typedef struct {

unsigned char ipraw[MAX\_SS7\_MSG\_LEN] ;

unsigned int iprawlen ;

} SS7Message\_ip ;

1. **SS7Message\_M2PA**

typedef struct {

/\* MTP2, Raw message starting FSN, FIB, BSN, BIN \*/

unsigned char m3uaraw[MAX\_SS7\_MSG\_LEN] ;

unsigned int m3uarawlen ;

/\* MTP Layer 3 Variables \*/ unsigned int opc ;

unsigned int dpc ;

unsigned char message\_type ;

unsigned char si ;

unsigned char ni ;

unsigned char mp ;

unsigned char sls ;

} SS7Message\_m3ua

1. **SS7Message\_M3UA.**

typedef struct {

unsigned char message\_type ;

unsigned int bsn ;

unsigned char bib ;

unsigned int fsn ;

unsigned char fib ;

unsigned short li ;

unsigned char sio ;

} SS7Message\_m2pa ;

UserApplication will identify from the messages(m2pa,m3ua,ip) and will be writing the contents of that stream into File and Database. So that it can be used later if required as Historic Data.

Execution order will be :

1. SP Probe
2. UserApplication.

#### Output dependencies

UserApplication will store the message in File and Database after decoding the stream (message format) coming from SP Probe.

Files:

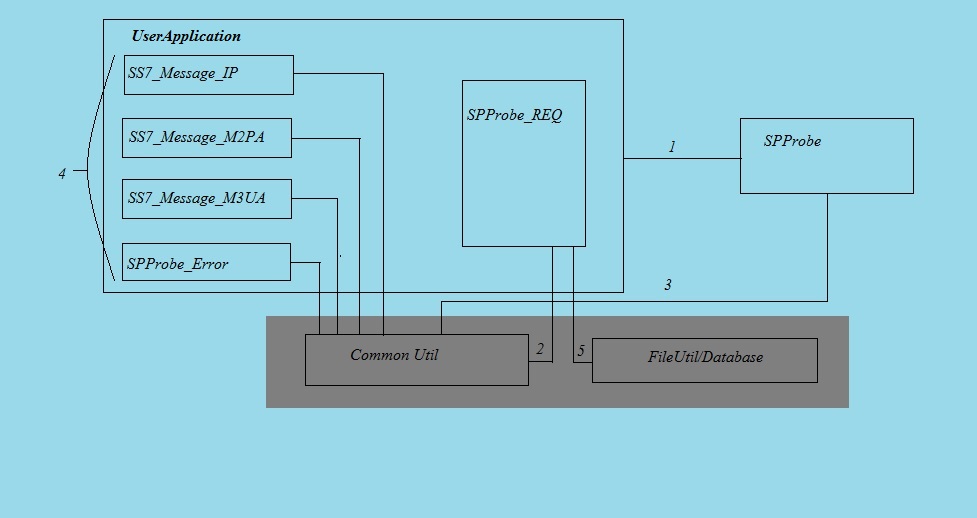
* IP\_MSG\_FILE.txt.
* M2PA\_MSG\_FILE.txt.
* M3UA\_MSG\_FILE.txt.

Database:

* SS7Message\_M2PA\_Table.
* SS7Message\_M3UA\_Table.
* SS7Message\_IP\_Table.

## Call Flows/Time Diagrams/Application & Process States

Call Flow:



1.UserApplication gets connection With SP Probe.

* Initial request send/receive from UserApp to SP Probe

UserApp (Initial request) SP Probe.

* Connection Established between SP Probe and UserApp.

SP Probe (Ack, Port) UserApp.

* Exchange of Port and IP Address between SP probe and UserApp.

UserApp (send Port, IP Address) SP Probe.

* Connection Established between SP Probe and UserApp.

SP Probe (getport (), getIPAddress ()) UserApp.

2. Communication Using CommonUtil

* Contains Type of Message (IP/M2PA/M3PA) send by SP Probe.
* Types of message are described in SPProbe\_REQ.

3.Read Type of message From SP Probe.

* i.e.(IP/M2PA/M3UA)

4.Set Type of messages in Com.Pojos package.

* SS7\_Message\_IP.
* SS7\_Message\_M2PA
* SS7\_Message\_M3UA
* SPProbe\_Error.

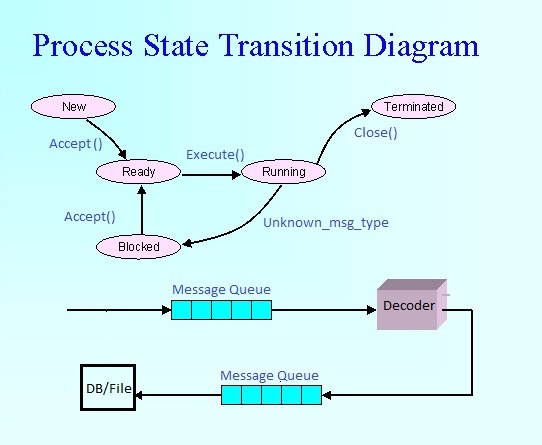
5.Write to particular File or Database or Other Application.

* FileUtil/Database.

Time Diagram



Application and Process States:



Whenever connection is established new process is created due which is handled by thread.

* Accept() creates new thread for every new Process(request).
* Execute() from the message queue data is fetched and read(decoded)
* Unknown\_msg\_type other unexpected message format send acknowledgment back to sending module informing to send correct format.
* close() request for closing of connection.

## Application/Module Interfaces and I/O Definitions

### Interfaces

* SNMP API

To send alarm message to SNMP Manager

* SP Probe API

To do communication with SP Probe Application

### Inputs

Can be any Message type from Mentioned below:

1. **SS7Message\_IP.**

typedef struct {

unsigned char ipraw[MAX\_SS7\_MSG\_LEN] ;

unsigned int iprawlen ;

} SS7Message\_ip ;

1. **SS7Message\_M2PA**

typedef struct {

/\* MTP2, Raw message starting FSN, FIB, BSN, BIN \*/

unsigned char m3uaraw[MAX\_SS7\_MSG\_LEN] ;

unsigned int m3uarawlen ;

/\* MTP Layer 3 Variables \*/ unsigned int opc ;

unsigned int dpc ;

unsigned char message\_type ;

unsigned char si ;

unsigned char ni ;

unsigned char mp ;

unsigned char sls ;

} SS7Message\_m3ua

1. **SS7Message\_M3UA.**

typedef struct {

unsigned char message\_type ;

unsigned int bsn ;

unsigned char bib ;

unsigned int fsn ;

unsigned char fib ;

unsigned short li ;

unsigned char sio ;

} SS7Message\_m2pa ;

### Outputs

UserApplication will store the message in File and Database after decoding the stream (message format) coming from SP Probe.

Files:

* IP\_MSG\_FILE.txt.
* M2PA\_MSG\_FILE.txt.
* M3UA\_MSG\_FILE.txt.

Database:

* SS7Message\_M2PA\_Table.
* SS7Message\_M3UA\_Table.
* SS7Message\_IP\_Table.

For Example File will contain the below mentioned format:

e.g.M3UA message

SS7Message\_M3UA [m3uaRaw=M3UA ISUP, m3uaRawLength=9, opc=10, dpc=10, messageType=M3UA\_Message\_type, si=M3UA\_SI, ni=M3UA\_NI, mp=M3UA\_MP, sls=M3UA\_SLS]

## Internals

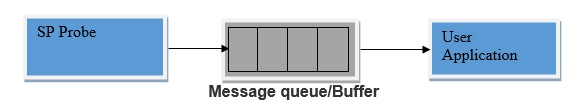
### Multi-Threading

* Get performance using multi-threading.
* Every request is handled by new thread.
* We monitor every thread and its events.

### Buffers/Queue Management

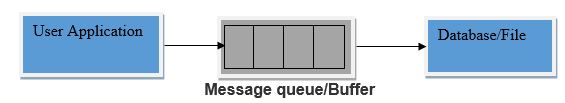
Threads are maintain to handle different type of Message Format it receive.

That are stored in message queue/Buffer and then received message will be handled.



Threads are maintain to handle different type of Message Format it receive.

That are stored in message queue/Buffer and then received message will be handled and send to Database/File.



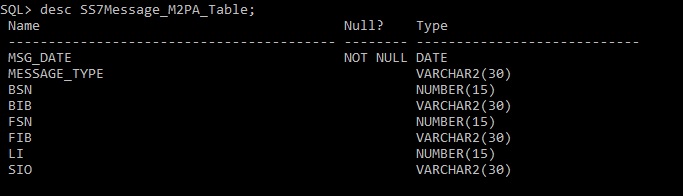
### Application Self-Health Monitoring

1. Proper threads are manage to handle every request(Type of message) coming from SP Probe Application.
2. Heart Beat is maintained for checking connection status between SP Probe Application and User Application.
3. Connection Status is checked between Database/File and User Application.
4. Loggers are maintained to keep tracking about events and identify Errors/Bugs or some unexpected outputs.
5. Alarms are maintained to signal the threats generated in User Application.

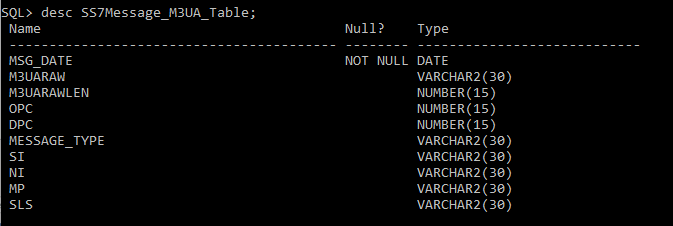
Before system gets crashed so that it can be handled soon.

### Database Design

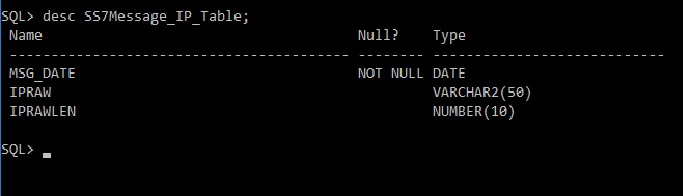
**1. Table SS7Message\_M2PA\_Table.**



**2. Table SS7Message\_M3UA\_Table**



**3. Table SS7Message\_IP\_Table;**



### Others Sources

* Internet
* Wikipedia

## Auxiliary/Platform Modules

### Configuration Management

We provide configuration functionality to configure our application using property file are as follows

* Configure User Application like changes in IP Address, Port and Modes.
* Configure Database ( Change the Database Url , User name ,Password)
* Configure Application Logger (Change log file path, name, rolling time)

### Logging

* Get log information on database, files and console.
* We use Apache Log4j in our application.log4j is highly configurable through external configuration files at runtime.

### Alarms and Notifications

* Get notification about application issue using SNMP agent
* We use Apache SNMP4J API to create SNMP agent.
* We get information about load, error and configuration.

Get/set request

SNMP

Manager

SNMP

Agent

Get/set response

### Reporting

//To Do

## Hardware and Software Environment Requirements

* **Recommended Server**

Any telecom grade server would be suitable for SIGTRAN Monitoring along with an add-on NIC.

* **Recommended OS**

RHEL 6.0, Windows, OSX.

### Development Environment

|  |  |  |
| --- | --- | --- |
| Tools Type | Tool Name | Tool version |
| IDE | Eclipse JEE (Luna) | 4.4 |
| Database | Oracle | 11g |
| Technology | Java | 8.0 |
| Operating system | Windows | xp & above |
| SDK | Java SE Development Kit(JDK) | 7.0 & above |

### Testing Environment

|  |  |  |
| --- | --- | --- |
| Tools Type | Tool Name | Tool version |
| IDE | Eclipse JEE (Luna) | 4.4 |
| Database | Oracle | 11g |
| Technology | Java | 8.0 |
| Operating system | Windows and Ubuntu | xp & above |
| Runtime Environment | Java SE Runtime Env (jre) | 7.0 & above |

### Production Deployment Environment

|  |  |  |
| --- | --- | --- |
| Tools Type | Tool Name | Tool version |
| Server | Any |  |
| Database | Oracle | 11g |
| Technology | Java | 8.0 |
| Runtime Environment | Java SE Runtime Environment (jre) | 7.0 & above |

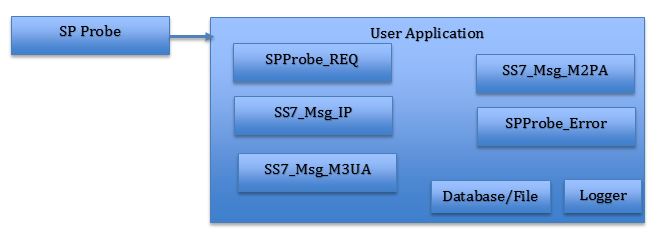
# High Level Test Plan and Testing Requirements

To do testing of user application we have to set all following tools

* Java Runtime Environment
* Jars log4j.jar , snmp4j.jar, Oracle Ojdbc6.jar
* One server (SP Probe app) who can message stream and it can register with user application. We just put our

IP address and port number in at the time of connection then server (SPProbe App) send acknowledgement to user app. Then user app receives message stream sent by SPProbe App.

# Considerations for Troubleshooting and Production Support (Sample Scenarios and work flow)



SP Probe: Provides input Stream (Message)

SPProbe\_REQ: Checks Stream Format and accordingly sends to respective

Modules.

SS7\_Msg\_IP: Module which handles IP messages format.

SS7\_msg\_M3UA: Module which handles M3UA messages format.

SS7\_msg\_M2PA: Module which handles M2PA messages format.

SPProbe\_Error: Handles unexpected unknown type of message format.

Database/File: Used for storing the Input Stream (Message)

Logger: Maintained to keep tracking about events and identify

Errors/Bugs or some unexpected outputs.

# To Do’s and Roadmap

* Application should be configurable using XML.
* Logger should get rotate after some size/Date and save the old log file.
* Changes made in xml should be read automatically without restarting application Using SNMP command.
* Logger put log info in file, console, Database