VPNS

MSE Mediation Service

This document contains material which is a proprietary property of and confidential to Verizon.

Disclosure outside Verizon is prohibited except by written permission, license agreement or other

confidentiality agreement. Unauthorized reproduction or distribution of this document in any

form or by any electronic or mechanical means is expressly prohibited.

Copyright ©2017 by Verizon. All Rights Reserved Worldwide.

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Name** | **Description of Change** | **Version Number** |
| 04/21/2018 | Madhu Repala | Created and released document | 2018.1 |
|  |  |  |  |
|  |  |  |  |

[1 Introduction 3](#_Toc516035540)

[2 Architecture 3](#_Toc516035541)

[3 Mediation Service Manager 3](#_Toc516035542)

[4 Alarm Receiver 3](#_Toc516035543)

[5 Jython Engine 4](#_Toc516035544)

[6 Alarm Requester 6](#_Toc516035545)

[7 Silent Detector 6](#_Toc516035546)

[8 Flood Controller 7](#_Toc516035547)

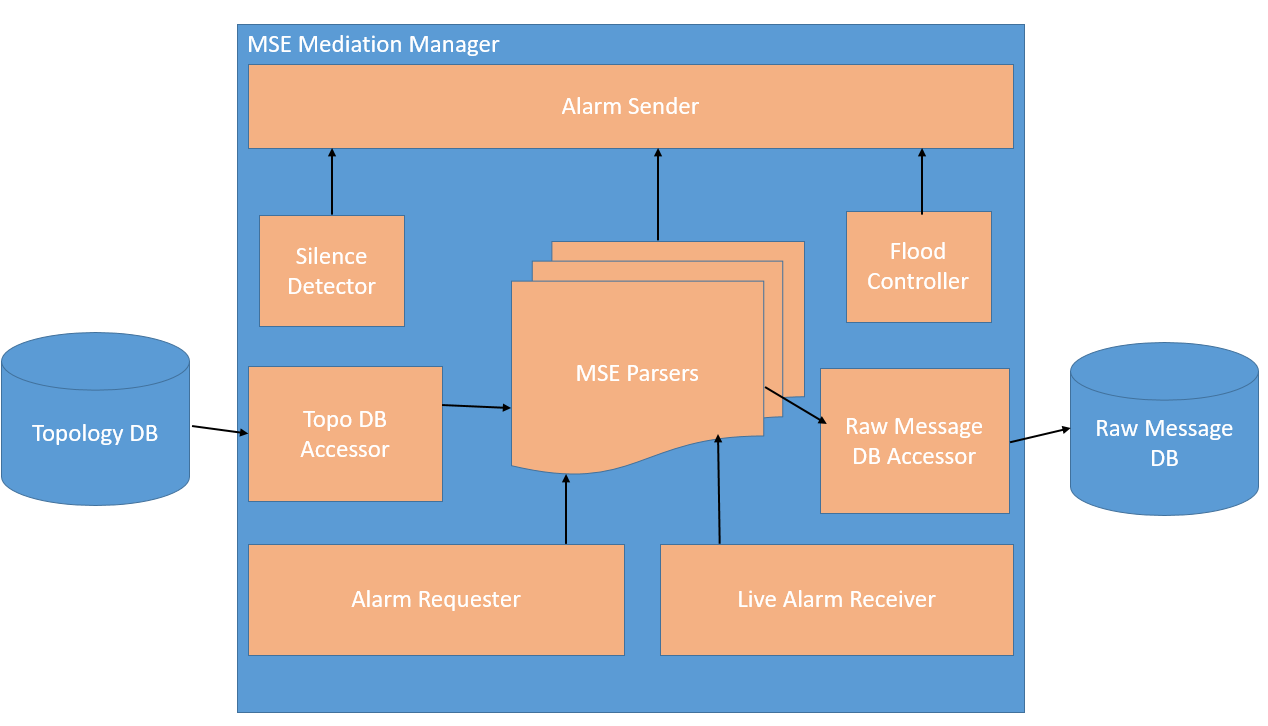
[9 Alarm Sender 7](#_Toc516035548)

# Introduction

The goal of this document is to provide Detailed Design Document for the Mediation Service of the MSE Network Elements. It provides detailed understanding of the code flow for the mediation, alarm querying, alarm resync and the live alarm receiver.

# Architecture

The following diagram depicts the high-level flow of the alarms from Alarm receiver to Alarm Sender.



# Mediation Service Manager

MSE Mediation Service provides a functionality for querying the existing alarms from the BNC subtended system through a REST Full web service. Also BNC system exposes a Web socket service to consume the live alarms feed.

# Alarm Receiver

During service startup, it establishes a web socket client connection with the BNC system and will keep receiving live alarms.

BNC system details like Name, IP Address and port details are retrieved from configuration file <config.par>

This functionality is implemented in the class MSEJettyWebSocketClient and it uses eclipse jetty web socket library *WebSocketAdapter* and implements all methods of websocket interface like openWebSocket, onWebSocketBinary, onWebSocketClose, onWebSocketConnect, onWebSocketError and onWebSocketText.

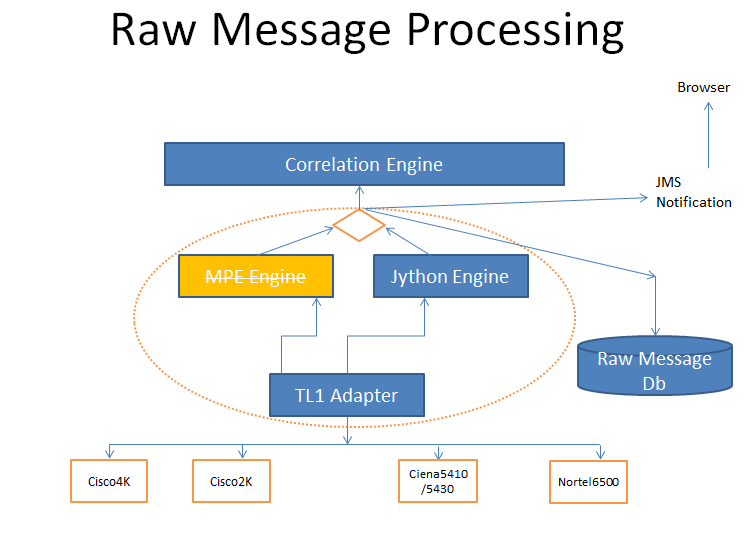
The JSON formatted alarms text is received in the method onWebSocketText and are kept in the queue.

A separate thread will be processing these alarms by sending them to MSE Parsers. These MSE parsers are implemented in python.

# Jython Engine

The Jython Engine, basically it parses the JSON string to the key, value pair and applies the alarm condition rules. Also it composes some fields based on the received parameters in the alarm. Those special fields are component\_id and Alarm Identifier.

These fields are key parts in the correlation engine which plays major role to find the correct inventory entity and post the alarms on to those entities.



Jython Engine uses a thread pool (configurable number of threads) to process the incoming TL1 Messages and it is designed in such a way that all the TL1 Messages of a particular TID gets processed by only one thread to maintain the order of the Alarm processing.

The actual processing logic is implemented using python scripting. Python script holds the alarm condition types which are supported by the VPNS/Mediation Manager. All the alarms

Sample Raw Alarm Message:

 {  
                "alarm": {  
                    "alarm-category": "ROUTING",  
                    "alarm-id": "",  
                    "alarm-text": "",  
                    "alarm-type": "CONDITION \_FAIL",  
                    "alarm-type-qualifier": "",  
                    "alarmed-object-name": "",  
                    "circuit-id": "",  
                    "generation-time": "2018-06-05T19:03:44Z",  
                    "interface-index": "",  
                    "neighbor-ip": "1.1.1.1",  
                    "neighbor-system-id": "",  
                    "node-id": "NODE\_ID",  
                    "node-ip": "1.1.1.1",  
                    "object-type": "MPLS\_TE",  
                    "port": "",  
                    "raw-alarm": “{}",  
                    "service-affecting": "",  
                    "severity": "MINOR",  
                    "shelf": "B1",  
                    "slot": "",  
                    "subport": "",  
                    "subslot": "",  
                    "subslot2": "",

Sample Alarm after Parsing:

After computing Fields

{severity=INFO, node-id=NODE\_ID, CONDTYPE=UI \_CMD, ATAG=2018-06-05T17:06:40-00:00, raw-alarm={"jet-event":{"event-id":"UI \_CMD","hostname":" NODE\_ID ","time":"2018-06-05-17:06:40","severity":"info","facility":"interact","process-id":48893,"process-name":"mgd","message":"UI \_CMD: User 'bnc' used client to run command 'get-config'","attributes":{"username":"bnc","command":"get-config"}}}, alarm-type=UI \_CMD, ISFILTERED=T, GENERATIONTIME=2018-06-05T17:06:40-00:00, DESC=UI \_CMD: User 'bnc' used client to run command 'get-config', ALARMSOURCE=JUNIPER, generation-time=2018-06-05T17:06:40-00:00, alarm-text=UI \_CMD: User 'bnc' used client to run command 'get-config', RAWMSG={"node-id": " NODE\_ID", "interface-index": "", "raw-alarm": "{\"jet-event\":{\"event-id\":\"UI \_CMD\",\"hostname\":\" NODE\_ID \",\"time\":\"2018-06-05-17:06:40\",\"severity\":\"info\",\"facility\":\"interact\",\"process-id\":48893,\"process-name\":\"mgd\",\"message\":\"UI \_CMD: User 'bnc' used client to run command 'get-config'\",\"attributes\":{\"username\":\"bnc\",\"command\":\"get-config\"}}}", "alarm-type": "UI\_NETCONF\_CMD", "slot": "", "generation-time": "2018-06-05T17:06:40-00:00", "subslot": "", "alarm-text": "UI \_CMD: User 'bnc' used client to run command 'get-config'", "circuit-id": "", "node-ip": "1.1.1.1", "service-affecting": "", "subslot2": "", "object-type": "", "subslot3": "", "neighbor-ip": "", "alarm-type-qualifier": "", "severity": "INFO", "alarmed-object-name": "", "alarm-id": "", "subport": "", "shelf": "", "port": "", "neighbor-system-id": "", "alarm-category": "interact"}, node-ip=1.1.1.1, CLLI= NODE\_ID, alarm-category=interact, AID=FILTERED, EQUIPCLASS=MSE, SEVERITY=4}

# Alarm Requester

Alarm Requester is a component inside the MSE Mediation Service Manager. This component is started during service startup and retrieves all MSE Network Elements from the Topology Database.

Alarm Requester component starts resyncing alarms from the BNC system. The resync mechanism is provide from the BNC system by exposing RESTFull Web Services.

The following is the payload for the resync REST query.

{

"bnc-alarm-query:input": {

"device-name":"NETWORKELEMENT\_TID",

"start-time": "2018-03-13T18:45:00-00:00",

"end-time": "2018-03-21T19:30:00+00:00"

}

}

This query results the alarms those are generated between the start-time and end-time. Once the alarms are received from the BNC Query, those alarms will be sent to the parser i.e. Jython Engine.

This Alarm requester component is implemented using Scheduled Executer ThreadPool framework and one scheduled job per one Network Element is submitted to the executor.

If any Network Element is failed to produce the result then, the component retries by re scheduling the same job.

The Alarm Requester is also designed to handle any new Network Element is added to the Inventory, if a manual reload executed from the console and if a the Network Element is failed before and now it comes online.

# Silent Detector

A Silent Detector is another component implemented in MSE Mediation Service which detects the Network Elements those are not actively receiving alarms with in the stipulated time period.

This component is a separate Time Thread which runs periodically and checks the NEs one by one if the stipulated time is crossed from the last alarm received time.

If any Network Element is detected as silent then a LOV (Loss of Visibility) alarm is generated by the Mediation Service. And if the same Network Element receives any alarm and as it is already declared as LOV then a clear LOV is generated by the Mediation Service.

To implement this multiple maps are defined to hold the NE to last received time and already LOV declared NEs. All the thresholds and time periods are configurable and they can be changed dynamically.

# Flood Controller

Flood Controller is another component implemented in MSE Mediation Service which is responsible for detecting event flood happening from any Network Element and bloc the live alarms from that Network Element.

This is implemented using a Timer Task which runs periodically and finds the number of alarms received from any Network Element is crossed the predefined threshold. If any Network Element is crossed that threshold with in the time period then an event flood will be declared and that Network Element will be marked as Flooded.

And in the next cycle of execution if the same Network Element receives the less number of alarms then the threshold limit within the time period then as it is already marked as Flooded it will generate a clear flood event. Then the alarms will be forwarded to the next level for processing.

To implement this multiple maps are defined to hold the NE to last flood detect time and already flood detected declared NEs. All the thresholds and time periods are configurable and they can be changed dynamically.

# Alarm Sender

Alarm Seder is a component that is inside the Mediation Service. It holds a queue where all the processed messages are stored. And a separate thread picks the alarms and sends them to the VPNS system for the correlation. Alarm sender does some enrichment of the alarm by populating some vpns required fields and translates the parsed messages into the java objects. After translating the alarms into java objects, it sends to VPNS by using CORBA interface.