**Functional Specification**

**For**

**Service Level Fault Management**

Current Revision: 1.4

Current Date: 11/10/2011

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date Issued | Description | Author |
| 1.0 | 10/06/2011 | Initial version | Jimmy Jin and Vibhu Pratap |
| 1.1 | 10/17/2011 | Add more design | Jimmy Jin |
| 1.2 | 10/27/2011 | Add SNMP trap details | Shaogang Chen |
| 1.3 | 10/29/2011 | Added Drools section  Added TOC | Paulo Sant’Anna |
| 1.4 | 11/10/2011 | Added/Updated Service APIs | Jimmy Jin and Paulo Sant’Anna |

This Functional Specification document has been approved and accepted by:

|  |  |
| --- | --- |
| Functional Specification Approvers | Date |
| (enter appropriate approval name & position) |  |
|  |  |
|  |  |
|  |  |
|  |  |

TOC

[TOC 3](#_Toc307577429)

[1. Introduction 4](#_Toc307577430)

[2. Requirement 7](#_Toc307577431)

[3. User Stories 9](#_Toc307577432)

[4. High Level Design 10](#_Toc307577433)

[5. Drools 17](#_Toc307577434)

# 

# Introduction

This document describes the requirement and high level design of supporting service level fault management. The main functionality of this feature is to use open NMS infrastructure to correlate device level fault events so that high level constructs in a Junos Space application can be monitored.

# Picture3

Although the design should be generic, the initial support will be focus on EABU’s service provision application, which configures network services such as point to point, point to multipoint and mesh services. Service types include, L2 VLAN and L3 VPN. An example of network service is illustrated as following diagram.

# Picture4

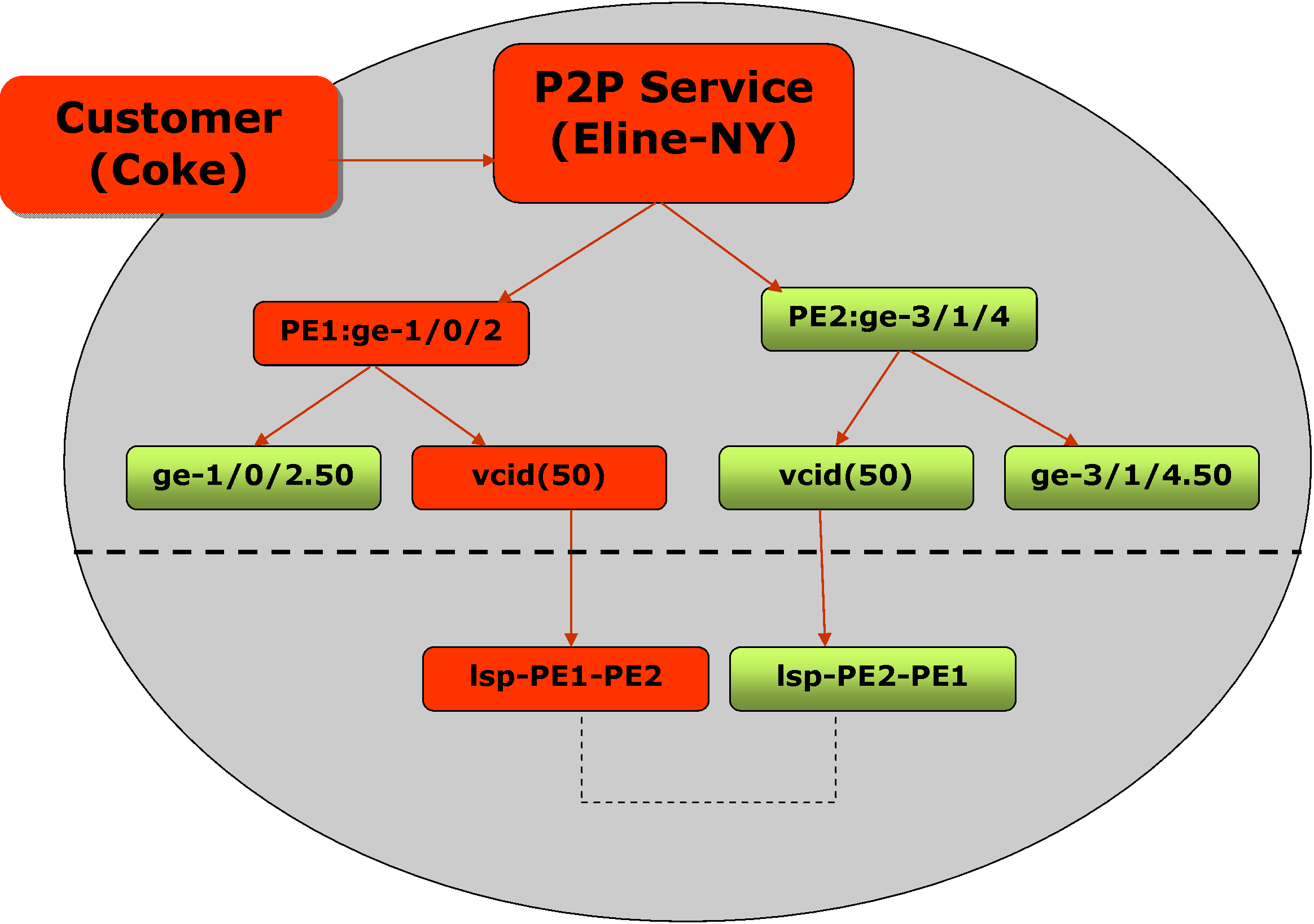
A service in this context is defined a network connectivity offered by service provider to end user. Enable a network service usually requires configuring multiple device and establishing several network segments. Each segment is called a service element.

# Picture6

A service instance of P2P is shown as:

# Picture7

When a device detect fault, lsp-PE1-PE2 down occurs and reported to Open NMS through SNMP trap, jnxLdpSesDown, an service level alarm should be raised by correlating lower level events.



**ge-1/0/2**

**ge-3/1/4**

# Requirement

# **Service Fault Correlation**

Requirement -1: When device level detectable fault occurs, corresponding service fault should be reported.

Note: the service elements forming a service are distributed among multiple devices.

Detectable from device

1 : N

ALL

1 : 1

1 : 1

ALL

N : 1

N :1

ANY

HLO Alarm raised by open NMS

N :1

Use case-1:

1. Receive UNI down trap OR Pseudo wire down trap,
2. Generate service element down alarm and service down alarm.

Use case-2:

1. Receive LSP-A down trap AND LSP-B down trap,
2. Generate transport down alarm,
3. Generate all Pseudo wires (on the transport) down,
4. Generate all service elements down alarm,
5. Generate all services down alarm.

Use case-3:

1. Receive physical interface down trap,
2. Generate all UNIs (on the physical interface) down,
3. Generate all service elements down alarm,
4. Generate all services down alarm.

# **Alarm Aggregation**

Requirement -2: All HLO alarms caused by the same root cause should be sent single event containing all services, when the alarm corresponding to that root cause is received.

Use case-4:

1. Receive a physical interface down trap;
2. Drools: all UNIs on that interface will be marked down, which causing all corresponding services marked down.
3. However, only single event should be sent to Space with all the services affected.

Use case-5:

1. Receive both LSP-A and LSP-B down traps;
2. Drools: all pseudo wires on that transport are down marked down, which causing all corresponding services marked down.
3. However, only single event should be sent to Space with all the services affected.

Requirement -3: Multiple HLO alarms should be aggregated and sent within one event, if the underneath device level traps/alarms are received within a time window. It is possible that these device level traps and alarms could be caused by the same root cause.

Use case-6:

1. Receive multiple UNI interfaces on the same physical interface are down within 30 seconds,
2. Send single service down alarm/event containing all the services affected to Space.

Use case-7:

1. Receive multiple pseudo wires on the same transport are down within 30 seconds,
2. Send single service down alarm/event to Space with all the services affected.

# **Modeling HLO Alarm in Open NMS**

Requirement 4: Service alarms should be modeled and displayed in the opennms, so user can browser the HLO alarms and drill down for details.( Note that modeling and show service itself is not a required for now.)

Use case -8

* GUI should display all service alarms, or filtered with service name/id;
* Drill down to what device event causing that alarm;

# **Space Platform Features**

Requirement -5: All HLO alarms should be sent to the Space Application using JMS.

Requirement-6: Platform provides APIs to add and delete a service.

Requirement-7: Support up to 15K services.

# **Enhancement**

**SNMP Community String**

Requirement -8: If a device SNMP community has a non-default value (Public), Space should import the customized string into Open NMS. If device is using SNMP v3, user name and password need to be configured in Open NMS as well.

**User Management**

Requirement -9: When a Space user is added, the user with the same password should be added into Open NMS user list. (Redirect to Open NMS UI for inputting user attributes is desirable)

**Event Retention Policy**

**Requirement -10: Define a default event persistent policy, so that the event/alarm table is maintained in a reasonable size.**

**What need to define is:**

1) If the trap is unnecessary to the system, it can simply be discarded by Trapd before it is published as an Event.  To do this, change the event configuration and in the logmsg.dest attribute set "discardtrap" as the value.

<logmsg dest="discardtrap" />

2) If you want the Trap to be published as an Event for the rest of the system to see but don't want it in the DB (and therefore not in the EventList), you can set the logmsg.dest attribute to "donotpersist".

<logmsg dest="donotpersist" />.

3) If you want the Trap to be published and persisted but you only want to retain the most recent occurrence, then activate it as an alarm and set the auto-clean attribute to "true"

<alarm-data auto-clean="true" />

# User Stories

*<List all user stories associated with this functional area. Only include the user stories that are visible to the customer through the UI.>*

|  |  |  |
| --- | --- | --- |
| **User Story** | **Statement** | **Acceptance Criteria** |
| PL-452 | Service Level Fault Management |  |
|  |  |  |
|  |  |  |
|  |  |  |

# High Level Design

# **Module Interaction Diagram**

The following diagram shows interactions among related modules. Note that in order to keep down the number of facts in the drools working memory, service, service element and device faults model is inserted into the working memory when a device event is received, instead of when a service is deployed.

6. Insert dependency

5. fault event

3. Add service

2. Service deploy

**platform**

1. Install

4. Trap/ syslog

7. Service

down/up

Rules

device

**application**

model

**In the above diagram,**

* **Application register/plug in the rules in the rule file during the application installation time.**
* **When a service is provisioned and platform FMPM module will add service and its service elements into Open NMS for monitoring.**
* **When events from devices are received, if the device fault is part of the service element, Drools inserts dependency model and fire the rules.**
* **High level service impact event will be generated. The service level events can be display in Open NMS GUI or/and high level application GUI.**

# **Sequence Diagram**

High level interaction diagram among the modules:

GUI: filtered service alarm

JMS: service down/up

Event: device fault on/off

Trap: fault occurs/clear

Event: add/remove service

EJB call: add/remove service ()

NA

JS-FM

Drools

Device

opennms

Rule: insert/retract dependencies

Correlation: service down/up

# **XML Data Structure**

# The API between the applications and platform makes use of JAXB generated classes from the following XSD.

|  |
| --- |
| **XML Schema for Monitor Service** |
| <?xml version="1.0" encoding="UTF-8"?>  <!-- edited with XMLSpy v2006 rel. 3 sp1 (http://www.altova.com) by shaogang (Juniper Networks, Inc.) -->  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://junosspace.juniper.net/monitoring" targetNamespace="http://junosspace.juniper.net/monitoring" elementFormDefault="qualified" attributeFormDefault="unqualified">  <xs:element name="MonitorableService">  <xs:complexType>  <xs:sequence>  <xs:element name="Id" type="xs:string"/>  <xs:element name="Name" type="xs:string"/>  <xs:element name="Status" minOccurs="0"/>  <xs:element ref="MonitorableServiceElement" maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="MonitorableServiceElement" type="MonitorableServiceElementType"/>  <xs:element name="MonitorableServiceElementComponent" type="MonitorableServiceElementComponentType">  <xs:annotation>  <xs:documentation>Comment describing your root element</xs:documentation>  </xs:annotation>  </xs:element>  <xs:complexType name="VariableBindingType">//  <xs:sequence>  <xs:element name="Key" type="xs:string"/>  <xs:element name="Value" type="xs:string"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="MonitorableServiceElementComponentType">  <xs:sequence>  <xs:element name="Type" type="xs:string" minOccurs="0">  <xs:annotation>  <xs:documentation>The MonitorableServiceElementComponent is a combination of several SNMP traps that indicate the status of the same entity</xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="Status" type="xs:string" minOccurs="0"/>  <xs:element name="Attribute" type="VariableBindingType" minOccurs="0" maxOccurs="unbounded"/>  <xs:element name="LogicalOperation" type="xs:string" minOccurs="0"/>  <xs:element ref="MonitorableServiceElementComponent" minOccurs="0" maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="MonitorableServiceElementType  ">  <xs:sequence>  <xs:element name="Id" type="xs:string"/>  <xs:element name="DeviceId" type="xs:string"/>  <xs:element name="Status" minOccurs="0"/>  <xs:element name="ServiceElementOperation" type="xs:string" minOccurs="0"/>  <xs:element ref="MonitorableServiceElementComponent" minOccurs="0" maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  </xs:schema> |

There are only two types of “ServiceElementOperation” in MonitorableServiceElementType: ADD, DETELE and MODIFY. A MonitorableServiceElementType with modification operation contains the new data; the old data with the same Id is wiped out.

A sample XML data with ADD operation is list below:

|  |
| --- |
| **Sample ADD Operation** |
| <?xml version="1.0" encoding="UTF-8"?>  <!--Sample XML file generated by XMLSpy v2006 rel. 3 sp1 (http://www.altova.com)-->  <MonitorableService xmlns="http://junosspace.juniper.net/monitoring" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://junosspace.juniper.net/monitoring  MonitorableServiceElement.xsd">  <Id>123</Id>  <Name>CokeP2P</Name>  <MonitorableServiceElement>  <Id>1234</Id>  <DeviceId>1234</DeviceId>  <ServiceElementOperation>ADD</ServiceElementOperation>  <MonitorableServiceElementComponent>  <LogicalOperation>AND</LogicalOperation>  <MonitorableServiceElementComponent>  <Type>jnxVpnPwUpDown</Type>  <Attribute>  <Key>jnxVpnPwVpnType</Key>  <Value>5</Value>  </Attribute>  <Attribute>  <Key>jnxVpnPwVpnName</Key>  <Value>ge-1/0/6.2</Value>  </Attribute>  </MonitorableServiceElementComponent>  <MonitorableServiceElementComponent>  <Type>jnxVpnIfUpDown</Type>  <Attribute>  <Key>jnxVpnIfVpnType</Key>  <Value>5</Value>  </Attribute>  <Attribute>  <Key>jnxVpnIfVpnName</Key>  <Value>ge-1/0/6.2</Value>  </Attribute>  </MonitorableServiceElementComponent>  </MonitorableServiceElementComponent>  </MonitorableServiceElement>  </MonitorableService> |

A sample XML data with DELETE operation is list below:

|  |
| --- |
| **Sample DELETE Operation** |
| <?xml version="1.0" encoding="UTF-8"?>  <!--Sample XML file generated by XMLSpy v2006 rel. 3 sp1 (http://www.altova.com)-->  <MonitorableService xmlns="http://junosspace.juniper.net/monitoring" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://junosspace.juniper.net/monitoring  MonitorableServiceElement.xsd">  <Id>123</Id>  <Name>CokeP2P</Name>  <MonitorableServiceElement>  <Id>String</Id>  <DeviceId>1234</DeviceId>  <ServiceElementOperation>DELETE</ServiceElementOperation>  </MonitorableServiceElement>  </MonitorableService> |

The expected response is shown below:

|  |
| --- |
| **Sample Response** |
| <?xml version="1.0" encoding="UTF-8"?>  <!--Sample XML file generated by XMLSpy v2006 rel. 3 sp1 (http://www.altova.com)-->  <MonitorableService xmlns="http://junosspace.juniper.net/monitoring" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://junosspace.juniper.net/monitoring  MonitorableServiceElement.xsd">  <Id>123</Id>  <Name>CokeP2P</Name>  <Status>Down</Status>  <MonitorableServiceElement>  <Id>1234</Id>  <DeviceId>1234</DeviceId>  <Status>Down</Status>  <MonitorableServiceElementComponent>  <Status>Down</Status>  <LogicalOperation>AND</LogicalOperation>  <MonitorableServiceElementComponent>  <Type>jnxVpnPwUpDown</Type>  <Status>Down</Status>  <Attribute>  <Key>jnxVpnPwVpnType</Key>  <Value>5</Value>  </Attribute>  <Attribute>  <Key>jnxVpnPwVpnName</Key>  <Value>ge-1/0/6.2</Value>  </Attribute>  </MonitorableServiceElementComponent>  </MonitorableServiceElementComponent>  </MonitorableServiceElement>  </MonitorableService> |

# **Service Monitoring APIs**

Platform APIs are provided by fmpm-interfaces and the XML documents complies with the above XSD:

|  |
| --- |
| **Service Monitoring APIs** |
| ../FMPM/fmpm-interfaces:  public interface ServiceMonitoringInterfaces {  public ServiceId addNetworkService (String serviceModelXML);  public String deleteNetworkService (ServiceId);  public String addNetworkServiceElements (String serviceElementsXML);  public String deleteNetworkServiceElements (String serviceElementsXML);  }  Public interface OpenNMSInterfaces {  public void openNMSMessageRelay(String message);  } |

# **JMS Message for Service Level Alarm**

# A final requirement is that a Service impact can generate a JMS message to the application that registered the service. OpenNMS (Drools) send alarm notification by calling restful API (<https://localhost/fmpm/restful/NotificationMessageRelay>) of Space platform with the following XML:

|  |
| --- |
| **Alarm Notification** |
| <?xml version="1.0" encoding="UTF-8"?><!-- edited with XMLSpy v2006 rel. 3 sp1 (http://www.altova.com) by Juniper Networks, Inc. --><xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://junosspace.juniper.net/monitoring" targetNamespace="http://junosspace.juniper.net/monitoring" elementFormDefault="qualified" ><xs:complexType name="MonitorableServiceStatus"><xs:sequence><xs:element name="Id" type="xs:string"/><xs:element name="Name" type="xs:string"/><xs:element name="Status" type="xs:string"/></xs:sequence></xs:complexType><xs:element name="ServiceAlarmNotification"><xs:complexType><xs:sequence><xs:element name="ServiceAlarm" type="MonitorableServiceStatus" minOccurs="1" maxOccurs="unbounded"></xs:element></xs:sequence></xs:complexType></xs:element></xs:schema> |

# The JMS topic is defined in:

|  |
| --- |
| **Service Alarm JMS Topic** |
| $JMP-HOME/FMPM/system/sar-src/META-INF/hornet-jms.xml:<configuration xmlns="urn:hornetq"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"xsi:schemaLocation="urn:hornetq /schema/hornetq-jms.xsd"><!-- Topic definition --><topic name="NetworkServiceFaultNotification"><entry name="topic/NetworkServiceFaultNotification"/></topic></configuration> |

# **Related SNM Traps**

# **One monitorableServiceElementComponent type** is a combination of several SNMP traps that indicate the status of the same device entity. The following table shows the mappings.

|  |  |
| --- | --- |
| **MonitorableServiceElementComponent Type** | **SNMP Trap Name** |
| JnxVpnPwUpDown | jnxVpnPwUp NOTIFICATION-TYPE  OBJECTS { jnxVpnPwVpnType, jnxVpnPwVpnName, jnxVpnPwIndex }  STATUS current  DESCRIPTION  "A jnxVpnPwUp notification is generated when the Pseudo-Wire  with index jnxVpnPwIndex belonging to the VPN named jnxVpnPwVpnName  of type jnxVpnPwVpnType transitions out of the 'down' state."  ::= { jnxVpnMIBNotifications 3 }  jnxVpnPwDown NOTIFICATION-TYPE  OBJECTS { jnxVpnPwVpnType, jnxVpnPwVpnName, jnxVpnPwIndex }  STATUS current  DESCRIPTION  "A jnxVpnPwDown notification is generated when the Pseudo-Wire  with index jnxVpnPwIndex belonging to the VPN named jnxVpnPwVpnName  of type jnxVpnPwVpnType transitions to the 'down' state."  ::= { jnxVpnMIBNotifications 4 } |
| JnxVpnIfUpDown | jnxVpnIfDown NOTIFICATION-TYPE  OBJECTS { jnxVpnIfVpnType, jnxVpnIfVpnName, jnxVpnIfIndex }  STATUS current  DESCRIPTION  "A jnxVpnIfDown notification is generated when the interface  with index jnxVpnIfIndex belonging to the VPN named jnxVpnIfVpnName  of type jnxVpnIfVpnType transitions to the 'down' state."  ::= { jnxVpnMIBNotifications 2 }  jnxVpnIfUp NOTIFICATION-TYPE  OBJECTS { jnxVpnIfVpnType, jnxVpnIfVpnName, jnxVpnIfIndex }  STATUS current  DESCRIPTION  "A jnxVpnIfUp notification is generated when the interface  with index jnxVpnIfIndex belonging to the VPN named jnxVpnIfVpnName  of type jnxVpnIfVpnType transitions out of the 'down' state."  ::= { jnxVpnMIBNotifications 1 } |
| LinkUpDown | linkDown NOTIFICATION-TYPE  OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }  STATUS current  DESCRIPTION  "A linkDown trap signifies that the SNMP entity, acting in  an agent role, has detected that the ifOperStatus object for  one of its communication links is about to enter the down  state from some other state (but not from the notPresent  state). This other state is indicated by the included value  of ifOperStatus."  ::= { snmpTraps 3 }  linkUp NOTIFICATION-TYPE  OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }  STATUS current  DESCRIPTION  "A linkUp trap signifies that the SNMP entity, acting in an  agent role, has detected that the ifOperStatus object for  one of its communication links left the down state and  transitioned into some other state (but not into the  notPresent state). This other state is indicated by the  included value of ifOperStatus."  ::= { snmpTraps 4 } |

# **The LSP traps are monitored and would possibly trigger a service level Alarm in a later release:**

1. jnxLdpSesDown

# jnxLdpLspUp

# mplsLspPathDown

# mplsLspInfo<!-Up

# Drools

JBoss Drools will be used to implement the business logic for the service level fault management.

# **Facts**

OpenNMS needs to be enhanced to support the service-level objects below to be inserted as facts:

* Service: Comprised of ServiceElements, a Service can span multiple nodes
* ServiceElement: Comprised of ServiceElementComponents, a ServiceElement is deployed on specific node.
* ServiceElementComponent: The unit of monitoring. Events (Traps or Syslog messages) will have a 1-1 relationship with a ServiceElementComponent

The following Drool facts describe the fields of interest from the Drools perspective, OpenNMS will provide an API to map the objects into these facts, in similar fashion as already done for the Event object:

// Facts that match OpenNMS internal objects

**declare** Service

Id : String

Name : String

Status : String

// Any ServiceElement outage implies a Service outage (DependsOnAll behavior)

ServiceElementList : ArrayList // ArrayList<ServiceElement>

**end**

**declare** ServiceElement

Id : String

DeviceId : Long

Status : String

//Operation : String // This is only to ADD|DELETE|MODIFY the element in the DB, no need to persist

ComponentGroup : ServiceElementComponentGroup

**end**

**declare** ServiceElementComponentGroup

LogicalOperation : String // AND | OR : Controls how an impacted underlying ServiceElementComponent

// affects the ServiceElement itself:

// AND: The default, will map to the DependsOn fact

// OR: Will map to DependsOnAll fact

ServiceElementComponentList : ArrayList // ArrayList<ServiceElementComponent>

**end**

**declare** ServiceElementComponent

Name : String // Type in the XSD

AttributeList: ArrayList // ArrayList<ServiceElementComponentAttribute>

**end**

**declare** ServiceElementComponentAttribute

Key : String

Value : String

**end**

# **Dependencies**

The dependency facts will allow us to record the relationship between an individual ServiceComponent outage/impact and the ServiceElement that contains it. It is controlled by the LogicalOperation attribute in the ServiceElementGroup. There are two models:

* AND Model: This is the default. It means that the ServiceElement depends on ALL its ServiceElementComponents, so if any is affected then the ServiceElement is also affected
* OR Model: This allows for an individual ServiceElementComponent to be impacted and the ServiceElement itself will not be impacted, as long as at least one of the ServiceElementComponents is still available. For example if you have multiple interfaces creating an aggregated link, a single interface failure will not

bring the entire link down.

The following Drool facts will provide the mapping:

// Dependencies

// We have two dependency types, DependsOn and DependsOnAll

// DependsOn: The simplest type, if the dependency is impacted then the

// object is impacted

**declare** DependsOn

a : Object

b : Object

**end**

// DependsOnAll: The object is only impacted if ALL dependencies are impacted

**declare** DependsOnAll

a : Object

list : ArrayList

**end**

**NOTE:** The Service is always dependent on ALL it’s ServiceElements, so if any ServiceElement is impact the Service is automatically impacted.

# **Dynamic Fact Insertion**

To keep the fact footprint small, Service, ServiceElement and ServiceElementComponent facts will be inserted on demand based on events received. The sample rule below illustrates how an fact insertion could be triggered by an event:

// This rule will insert all facts relevant to a received event

**rule** "Insert Service Facts for JnxVpnIfDown Event"

**salience** 5000

**when**

$e : Event( uei **matches** ".\*JnxVpnIfDown", $nodeid : nodeid, $parms : parmCollection )

Parm( parmName == "jnxVpnIfVpnName", $vpn\_value: value ) **from** $parms

$vpn\_name : Value( $content : content ) **from** $vpn\_value

Parm( parmName == "jnxVpnIfIndex", $ifindex\_value: value ) **from** $parms

$ifindex : Value( $content : content ) **from** $ifindex\_value

**then**

// Insert Service(s), ServiceElement(s) and ServiceElementComponents(s) that include these attributes

// Could be implemented as a cascade of rules, for example, first only insert the ServiceElementComponents and

// then have other rules to find the ServiceElement that uses that component and insert that to, and then do the

// same for the Service that uses that ServiceElement

insertAll( "jnxVpnIfVpnName", $vpn\_name, "jnxVpnIfIndex", $ifindex );

# **end**