Towards Automated Interactions between the Internet and the Carrier-Grade Management Ecosystems

Small or medium scale focused research project (STREP) Co-funded by the European Commission within the Seventh Framework Programme
Grant Agreement no. 258300

Strategic objective: **The Network of the Future (ICT-2009.1.1)**

Start date of project: **September 1st, 2010 (36 months duration)**

Deliverable 3.3: "Final Report on Functional Design of the Basic and Advanced Modules in ONE" (M30)

Due date:	02/28/2013
Submission date:	03/01/2013
Deliverable leader:	UPC
Author list:	UPC: Marcelo Yannuzzi, Wilson Ramírez, Anny Martínez, Eva Marín-Tordera, René Serral-Gracià, Xavi Masip-Bruin TID: Víctor López, Óscar González de Dios, Amanda Azañon ADVA: MaciejMaciejewski, Kamil Kaluga, Christine Brunn TUBS: Marek Drogon, Mohit Chamania, Admela Jukan SNU: Jörn Altmann, Mohammad Hassan

Dissemination Level

☒	**PU:**	Public
☐	**PP:**	Restricted to other programme participants (including the Commission Services)
☐	**RE:**	Restricted to a group specified by the consortium (including the Commission Services)
☐	**CO:**	Confidential, only for members of the consortium (including the Commission Services)

Table of Contents

Figure Summary

1 Executive Summary

This document expands upon the functional design of the various modules of the ONE adapter as reported in the previous deliverable D3.2. Based on the implementation experiences in work package 4, this document defines interface specification for a large portion of the interaction between modules in the ONE adapter.

For the existing core and auxiliary modules, the document detailed functional design specification, building upon and updating the work in D3.2, and also provides detailed interface specification for interaction between the modules which is critical to the implementation activities in Work Package 4. The implementation experiences since D3.2 have also led to the modification of the functional design, such as the interactions between the workflow processor and the management controller which have been presented in this document. The document also updates the description of initiation operations and orchestration for various use cases within the scope of the ONE project based on the initial implementation of the same in WP4.

In keeping with the upcoming trend of Software Defined Networking (SDN), the document also includes a new functional module, namely the OpenFlow Control module, which can be used to interact with OpenFlow controllers and devices across multiple network layers. The document also presents discussion and interface specification for administration of the ONE adapter. Based on our implementation experience, the administration policy has been completely overhauled from the proposal in D3.2, and instead of a centralized entity, administration functions are now proposed at a per-module level, while AAA-based functionality has been introduced to ensure authorization of administrative operations.

The functional description presented in this deliverable indicates the completion of the scheduled milestones MS17 (Design of Advanced Modules).

2 Introduction

This deliverable provides a final report on the functional design of the basic and the advanced modules within the ONE adapter's architecture in compliance with the milestone MS17 scheduled in month 30 of the project's duration. The work reported here addresses the requirements and use cases that have been identified and developed in WP2, and is built upon the architectural definitions and design proposed inD2.2. The functional descriptions presented in this document extend in several ways the ones previously reported in deliverable D3.2. The document expands on the functional design presented in D3.2 based on the feedback from the implementation activities in WP4, and uses the implementation experience to provide interface specification for the different modules within ONE. The document also includes administration functions for modules within ONE, which provide the capability to configure the various features of the ONE adapter. The functional design presented here presents the design of all the core and auxiliary modules, as well as two advanced modules, namely the PCE and the AAA module. Finally, taking into cognizance the emerging SDN trend, the document also presents the design of the Openflow control module, which is designed to interact with an external controller to configure Openflow [OpenFlow] enabled devices.

As mentioned before, the functional specifications provided have been revised and improved based on our experience with implementing the same in WP4, and as a result the functional and interface specification have been have been provided in significant detail. This report provides pseudo-code specifications, flow charts, interface and parameter specification, and realistic examples of orchestrations, which in some cases touch the ground and mention specific technologies, interfaces, and protocols. The reason for this is to ease the tasks of WP4, but it is worth emphasizing that the designs described here are constrained neither by specific technologies nor by any programming framework or software. In other words, our designs are general in scope and can be perfectly supported by different implementations.

In order to be consistent with the original operation strategy of the ONE adapter, which is ***to receive triggers and execute workflows***, we have kept the core functionality independent from any external actor present in the network. In other words, the design of the core as well as the internal communication and data models used in the ONE adapter is agnostic of the technologies present in the operators' networks. It is also important to emphasize that the proposed adapter can be designed and implemented on the basis of already standardized interfaces and protocols as well as standardized workflow description and execution languages. With this approach, we intend not only to reduce time-to-market but also to ease the way for operators for adopting the ONE adapter.

The rest of this document is organized as follows. Section 3 revisits and describes in more detail the set of possible initialization actions for the use cases described in Deliverable D2.1. Section 4 addresses the functional design of the core modules. Section 5 covers the functional design of the basic and advanced "auxiliary" modules. Section 6 provides describes the different protocols that have been employed by modules in the ONE adapter to communicate with external systems and Section 7 outlines the main interactions of these modules in the context of the use cases identified in

Deliverable D2.1. Finally, Section 8 concludes this report. Detailed interface specification for all the modules presented can be found in the appendix (Section 12).

3 Initialization Actions

This section covers the actions triggering the execution of workflows within the ONE adapter. We describe the information contained in the triggers covering the parameters required for initiating the following set of basic workflows: i) IP link provisioning; ii) Service provisioning; iii) IP Offloading; and iv) Post failure recovery.

3.1 Triggering IP Link Provisioning

An IP link-provisioning request may be initialized either by a human operator (Operator-initiated triggers) or by an automated action (Automatically-Initiated). Operator-initiated triggers refer to triggers created by a network operator, e.g., through a graphical interface (that could be jointly provided with the ONE adapter), and will be received by the adapter's engine through Web Services. The Automatically-Initiated triggers are the triggers originated by an external programmable logic, such as an IP or a Transport Network Management System. An Automated-initiated trigger may come both in the form of Web Services. All the triggering actions in our architecture will be received by the Trigger Module (cf. Section 5.1) to be subsequently processed and executed in the form of workflows.

Of all the operations that can be coordinated by the ONE adapter, the IP link provisioning is the most basic one. In fact, this operation can be packed as an atomic workflow that could be repeatedly invoked and used either during a post-failure recovery or any IP service/offloading operation. This approach of sequentially using atomic workflows to compose larger ones (and there by perform more complex operations) along with the programmable nature of these workflows, are two of the most important features that shall be addressed in the functional design of the ONE adapter.

The information contained in the trigger that will be used for initializing the provision of an IP link may include (but it is not limited to) the following fields:

Field	***Description***
Mandatory Parameters	
Trigger ID	Trigger identifier.
AuthTicket	Authentication Ticket corresponding to a session that was initiated by a user (using the GUI) or automatically-initiated (by a NMS). The Authentication Ticket is used to confirm with the AAA if a certain session is valid.
Interfaces involved	Interface ID and basic requirements for each endpoint.
Addresses	IP address and subnet mask of each endpoint.
Next-Hop	IP address of the next-hop (in case of a static route configuration).
LSP	LSP type
Optional Parameters	

Duration	Applied only for temporary IP links
Bandwidth	Bandwidth required for each link.
Distance	Used for configuring an administrative distance. This parameter is just valid for Cisco routers.

3.2 Triggering IP Service Provisioning

Differently from the IP link provisioning case, an IP service-provisioning request may only be initiated by a network operator, i.e., it requires a human action to start the provisioning process. A GUI would enable the operator to select the provisioning actions needed, and transparently compose a request that could be sent to the ONE adapter using Web Services. The request will contain all the information needed by the ONE adapter to complete the IP service provisioning process, which may include (but it is not limited to) the following fields:

Field	***Description***
Mandatory Parameters	
Trigger ID	Trigger identifier.
AuthTicket	Authentication Ticket corresponding to a session that was initiated by a user (using the GUI) or automatically-initiated (by a NMS). The Authentication Ticket is used to confirm with the AAA if a certain session is valid.
Interfaces involved	Interface ID and basic requirements for each endpoint.
Addresses	IP address and subnet mask of each endpoint.
Service Type	VPN service, IPTV, Bandwidth on Demand, VoIP…
Service Mode	Unicast, Multicast…
LSP	LSP Type
Optional Parameters	
Duration:	Applied only for temporary services.
QoS Requirements	Jitter, delay, CBR…
Bandwidth:	Bandwidth required for the service
Routing	Routing policy (if required).

As mentioned above, the parameters contained in the above fields along with appropriate identifiers of the actions required to provide the desired IP service, could be selected (and even build or programmed) by the network operator in a friendly graphical environment, which would hide from the operator the complexity and the details of the configurations that will be needed at the Internet and transport layers. As we shall show in Section 5.1, these two pieces of information, combined with additional data and operator-defined policies stored in the adapter, will be used to build the triggers that will ultimately initialize the configurations and the coordination of management operations orchestrated through the ONE adapter.

It is worth highlighting that the **programmability** of the orchestrations performed by the ONE adapter is one of its main hooks, since this feature will allow operators managing multi-layer/multi-vendor networks to customize their provisioning, self-healing, and offloading operations in a coordinated fashion, and with the advantage of utilizing well-known and standardized interfaces and protocols (Web Services, SNMP, PCEP).The programmable nature of the orchestrations will also facilitate the evolution and adaptation of the adapter as new technologies and services come into the market.

3.3 Triggering IP Offloading

The IP offloading is a network status driven use case. This action to initiation IP Offloading is generated either manually (by a human operator based on the analysis of monitoring information), or automatically by monitoring and analyzing network information (e.g. reached threshold for link congestion).

In the case of manual IP offloading triggering, an authorized user (e.g. network administrator) is analyzing the outputs of the graphical user interface and notices increased traffic on a specific link or path. The user can create a new offloading event, which instructs the ONE adapter to offload the traffic onto an alternate (new) path.

In the automated IP offloading triggering, the ONE adapter can be configured to automatically create an event when the generated traffic increases beyond a pre-defined threshold and will attempt to offload this traffic across the core network using an optical circuit. When the application traffic increases beyond this threshold, the IP-NMS will be notified via a SNMP trap and the IP-NMS Event Listener will use this information to generate a new Event to initiate IP offloading.

After the Event is received, the ONE adapter will first identify the border routers associated with the application endpoints, and will check with the IP NMS if these routers have available interfaces to support a new optical circuit. The ONE adapter will then request the PCE to compute an optical circuit between these routers in the transport network. If the circuit is available, the ONE adapter will instruct the T-NMS to setup the computed circuit, and will then provide instructions to the IP NMS to configure the IP interfaces.

The trigger generated in this case will indicate the two endpoints required to perform the offloading, and a set of additional parameters, such as the capacity required, routing rules, etc., all of which may be included to initiate the offloading process.

The following fields should be filled to complete the trigger definition:

Field	***Description***
Mandatory Parameters	
Trigger ID	Trigger identifier.
AuthTicket	Authentication Ticket corresponding to a session that was initiated by a user (using the GUI) or automatically initiated (by a NMS). The Authentication Ticket is used to confirm with the AAA if a certain session is valid.
Interfaces involved	Interface ID and basic requirements for each endpoint.

Addresses	IP address and subnet mask of each endpoint.
Routing Rules	IP Routing rules to be configures to perform offloading operation
LSP	LSP type
Capacity	Capacity to be reserved
Optional Parameters	
Duration	Applied only for temporary IP links
Bandwidth	Bandwidth required for each link.
Distance	Used for configuring an administrative distance. This parameter is just valid for Cisco routers.

3.4 Triggering Post Failure Recovery

The “Post Failure Recovery” is, as the IP offloading case, a network status driven use case. This trigger will be generated as a result of network monitoring information received upon failures.

The basic goal in this use case is to initialize a sequence of operations through the ONE adapter with the aim of improving the survivability of the multi-layer network. For instance, new backup paths could be proactively created (or recomputed) in order to avoid risky situations such as double failures. Two actors can activate the Post Failure Recovery actions, namely, an operator through the GUI or the ONE adapter on itself by interpreting the notifications received from the monitoring system.

To this end, the trigger generated will indicate the endpoints required for performing the Post Failure Recovery actions, as well as extra parameters, such as the capacity that should be included to perform the Post Failure Recovery.

The following fields should be filled to complete the trigger definition:

Field	***Description***
Trigger ID	Trigger Identifier
Mandatory parameters	Router IDs. Capacity needed. Resilience degree.
Interfaces involved	Suggested interfaces to be used regarding the traffic origin and destination.
Duration	The time to keep activated the Post Recovery actions.
Optional Parameters	QoS parameters, etc.

4 Functional Design of the Core Modules

This section describes the functional design of the core modules within the ONE adapter architecture. We provide specific details on the inputs, outputs, internal workings and interactions for the Management Controller (cf. Section 4.1), the Ontology Mapper (cf. Section 4.2), and the Workflow Processor (cf. Section 4.3).

4.1 Management Controller Module

One of the main tasks of the *Management Controller* is to control the execution of workflows as instructed by the *Trigger Module* (refer to Section 5.1 for more details on the Trigger Module). To this end, in Figure 1, we present the different blocks forming the *Management Controller*. As it can be observed, this process is divided into three main blocks: Workflow Assignment, Smart Analytics, and Scheduler.

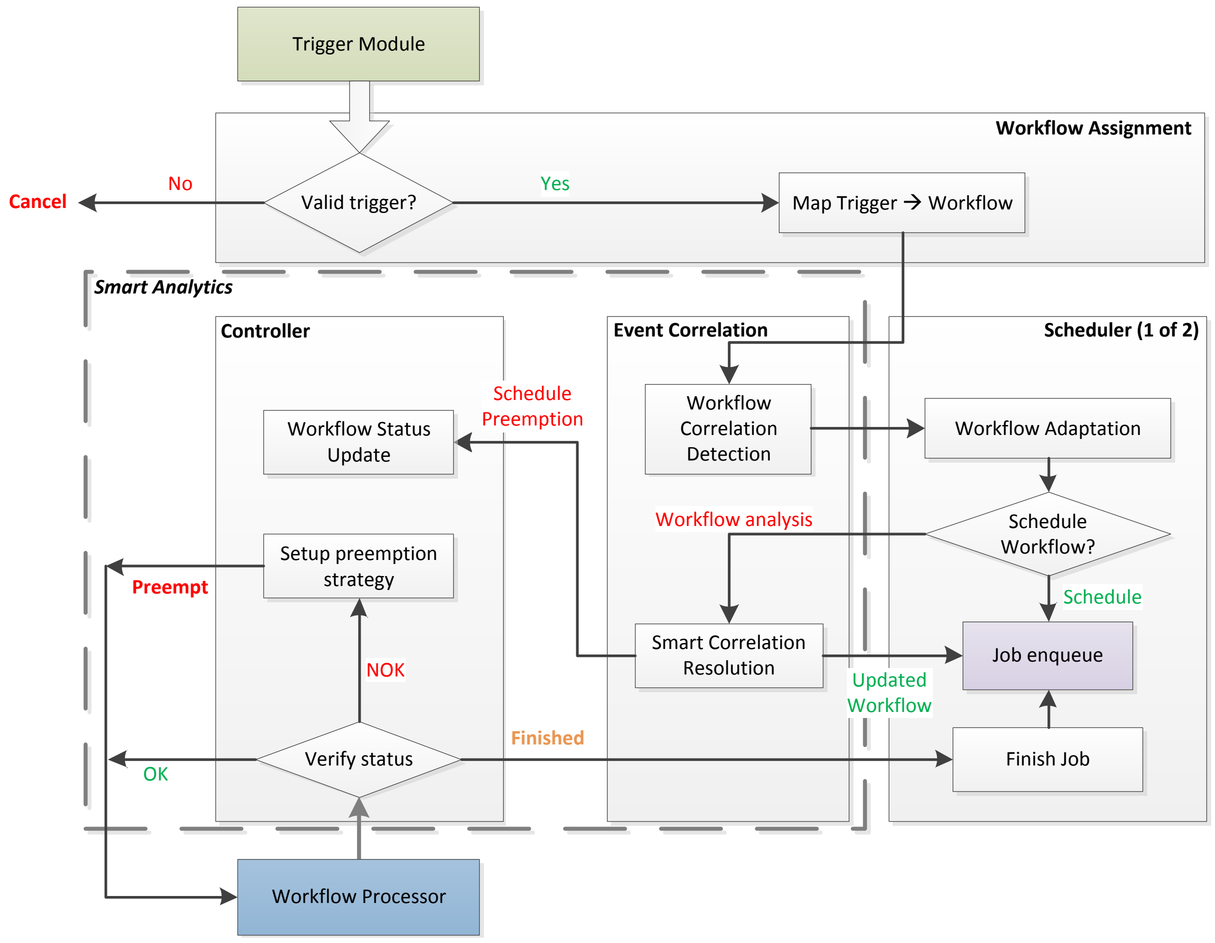


Figure 1. Overall interactions of the different parts that compose the Management Controller.

An important aspect of the *Management Controller* is that global status needs to be maintained, in order to effectively manage the different jobs and workflows in the system. The first consideration is that, internally, the *Management Controller* is devised as a job scheduler and smart analytics box, able to manage the resources according to the specific situations present in the system. To this end, internally, the *Management Controller* does not consider workflows or triggers as the management units but rather it considers **jobs**. A job is the unit which will be queued, executed, or pre-empted. The Workflow Assignment sub-module present within the MC will accept as an input a Trigger, and will output a job (which will contain the actual workflow plus internal management information, such as priority, “preempt ability”, and so on), then, the rest of sub-modules will take as input the job to be executed. This abstraction allows the management controller to seamlessly allow future extensions, and at the same time, generalize the different priorities and preemption strategies within the job. To this particular end, in Figure 2 we provide details on the global information required in the system.

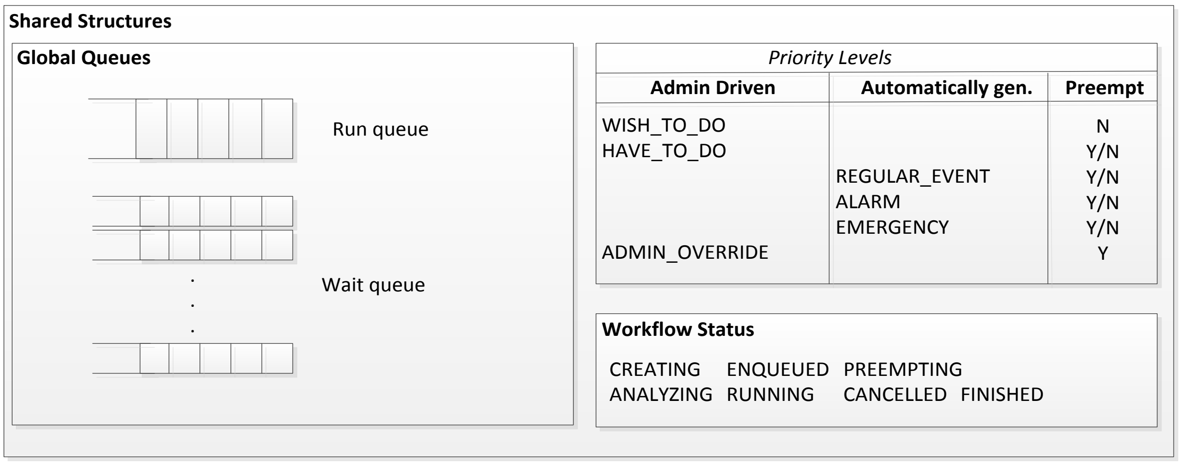


Figure 2. Global structures to define a Job

As we can notice from the figure, the priority levels are defined into two different categories: *administration driven* and *automatically generated*. The former stands for the jobs initiated by an administrator from the GUI, while the latter considers the jobs automatically generated by network events. Both priorities will be adapted by the Smart Analytics depending on predefined policies and the system status when the workflow enters the Management Controller. Within each category we define three different priority levels, each with its own preemption policy:

- **WISH_TO_DO**: This priority is intended for low priority events, triggered by the GUI.
- **HAVE_TO_DO**: Higher priority events triggered by the administrator. It allows preempting existing jobs up to ALARM priority.
- **REGULAR_EVENT**: Regular automatically generated event, for example IP offloading.
- **ALARM**: Important event, such as link down or self-healing.
- **EMERGENCY**: Serious event such as multiple link-down, or massive network disruptions.
- **ADMIN_OVERRIDE**: Very important job triggered by the GUI when it must override any running workflow or event.

The implementation of job management within the MC is performed through a state machine, as detailed in Figure 3. Initially a workflow will enter the system in the *Creating* state, once the required initializations are made, that is, it has been assigned a priority and a preemption policy, it must be analyzed to detect whether there is any conflict or correlation with the workflows already present in the system, either running or queued (*Analyzing* state). If there are no conflicts, the workflow will be scheduled (Q*ueued* state); otherwise it will be cancelled (*Cancelled* state). Once the workflow is queued, eventually it will be executed, moving to the *Running* state, or in the case of correlation with other workflows it could be also *Cancelled*. While running each workflow it will periodically validate its status, i.e., in the case that a higher priority workflow with the preemptive flag enabled enters the system, it will request that the current running workflow is preempted, thus, moving to the *Preempting* state, once the preemption has been finished, the workflow can potentially move to *Cancelled*, which implies that it is not necessary to run it anymore or it will be reanalyzed to be later re-q*ueued* in the system. In the case that the preempting workflow is a superset of the preempted one, it will move to *Finished*, as it does not need to be executed anymore. Another possible transition comes when a workflow successfully finishes its execution, moving from *Running* to *Finished* state.

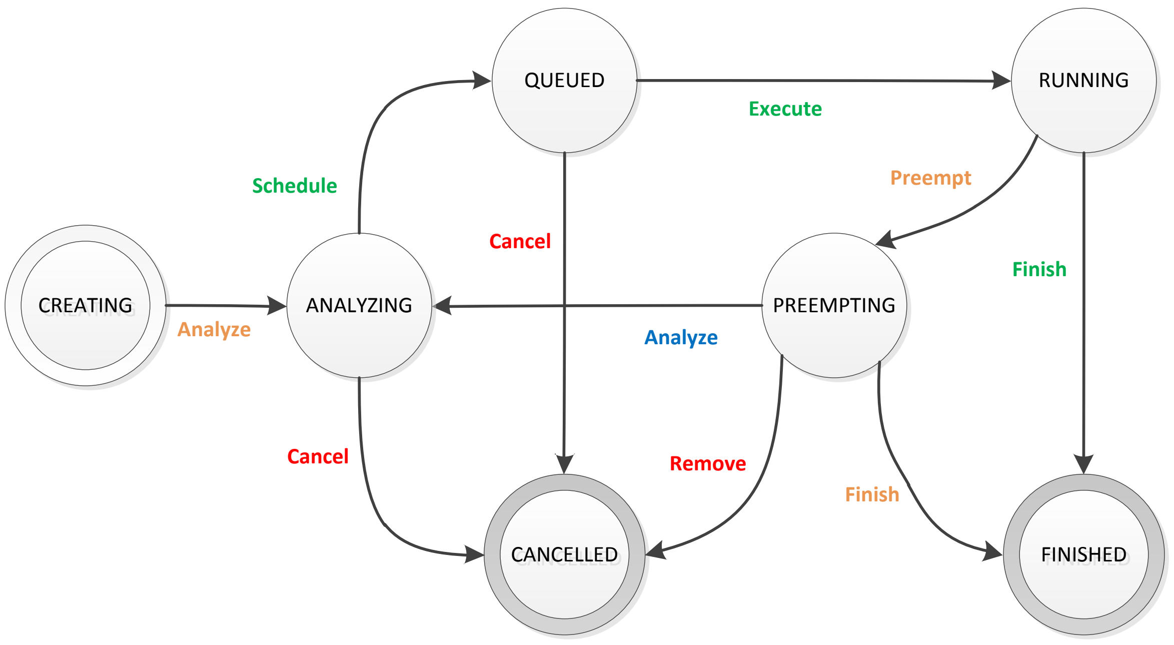


Figure 3. Different states of a Job within the Management Controller.

In the rest of this section, we will first detail the Workflow Assignment, the Controller, and the Scheduler. Afterwards, we will provide details on other important aspects of the *Management Controller*, such as, the Logging and Monitoring, and the Administration of ONE services.

4.1.1 Workflow Assignment

One of the primary functions of this sub-module is to check whether the workflow initiator (i.e., network administrator or device) is authorized or not to request the execution of a specific workflow. This function is accomplished along with the *AAA module* (refer to Section 5.8, for more on the AAA module). The main objective is to associate the execution of a workflow to a specific group of persons or devices. The network operator has to specify the association of people (i.e. network administrators) and devices to groups and the rights of groups, in advance. Figure 1 shows the *Workflow Triggering module* within the MC architecture. As shown in Figure 1, the Trigger Module (TM) sends the trigger information (trigger ticket, authentication ticket, and trigger parameters) to the MC (step 1). The workflow triggering sub-module checks with the *AAA module* whether the initiator is authorized or not to request the execution of the workflow as specified by the trigger information (step 2). If the *AAA module* approves the execution, then the WFT passes the received information to the Analytics sub-module (step 3). The Analytics module then initializes the workflow if the policy allows the workflow to be executed (step 4).

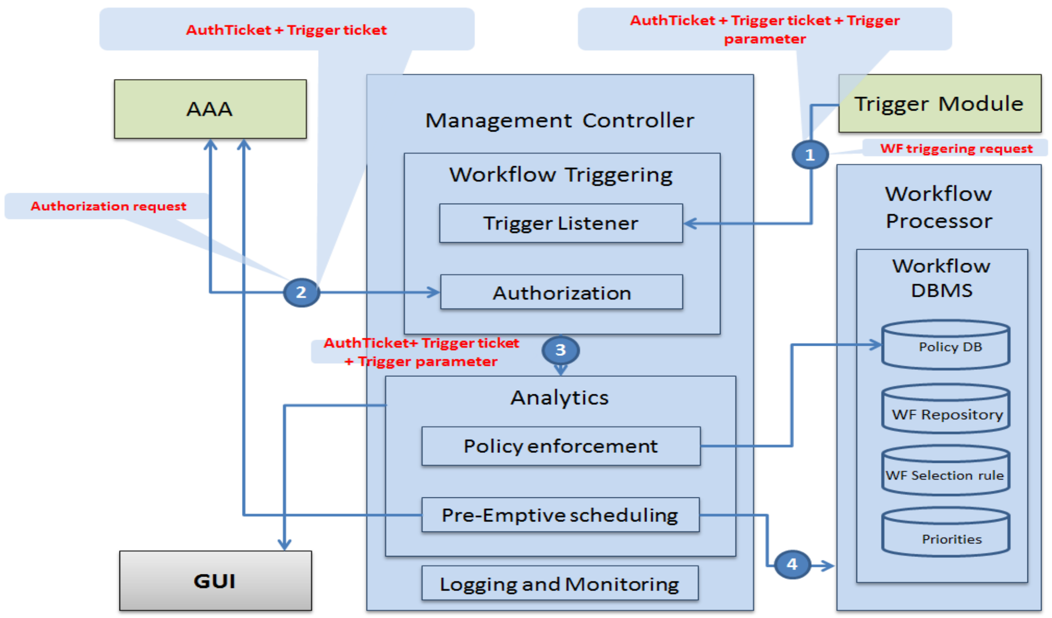


Figure 4 Workflow triggering (WFT) functional modules and its interfaces.

The interfaces that connect Workflow Triggering sub-module to the *Trigger Module*, Analytics sub-module, and AAA module are web services based.

The interface (1) between the Workflow Triggering sub-module and the *Trigger Module (cf. Section 12.2)* provides the means for receiving notifications on the request of a service, i.e., on the execution of a specific workflow.

The interface (2) to the *AAA module (cf. Section 12.12)* is used for authorization purposes. Through the Web Service interface, the WFT sub-module communicates with the *AAA module* to check if the workflow requester has the right to initiate such a workflow.

The Workflow Triggering sub-module has a web service interface (3) to the Analytics sub-module. This interface provides the means for exchanging all the information received through the *Trigger Module*, for further processing in the case that authorization is granted by the *AAA module* (refer to Appendix B, for more on WFT module interface definitions).

The Workflow triggering process can be described through the pseudo-code depicted in:

Table 1: Pseudo-Code for the Workflow Triggering Process

```
Input: Awaits the TriggerTicket(TT) & AuthTicket(AT) & TriggerParameter(TP)
Output: 0(Yes), 1(No), -1(Error)
        Wait for input from TM,
        If received:
                Pass the (TT) & (AT) to the AAA,

                Wait for AAA response,
                If the AAA reply is "Yes":
                        Pass the (AT) & (TT) & (TP) to the Analytics,
End (Return to "Await" status)
```

4.1.2 Smart Analytics

As we already stated on previous deliverables, the Smart Analytics sub-module is responsible of the decision making process, regarding the execution of workflows based on their priorities. To this end, it provides smart analysis of the incoming workflow and the ones being queued or executing, determining the required action in order to comply with the predefined policies within the adapter.

It is important to notice that this module does not enforce any changes to the executing workflow or to the wait queues in the *Management Controller*. But rather sets the priorities and pre-emption policies to the incoming trigger, identified as a job within the Smart Analytics sub-module, leaving all the queue management, job execution and pre-emption to the Scheduler.

To perform the above tasks the Smart Analytics sub-module is divided in two different parts, namely, Event Correlation and Controller, and it is tightly correlated with the Scheduler.

Job Correlation

As previously detailed in Figure 1 the job correlation is composed by the following parts:

- *Workflow correlation detection*: this part performs a first level of analysis of the incoming workflow. In particular, it analyses the relation of the workflow with the other workflows in the system. For example, it will evaluate aspects such as: Workflow duplication, Link Flapping (up/downs), and so on.

The main goal of the correlation detection is to quickly be able to decide whether a workflow can be executed or not, especially in the classical situation where no other workflows are present on the system.

Reads: The new Workflow, Run and Wait queues.
Writes: Job next status, Job Schedule, Job Cancel, or Job Analyze.

- *Smart Correlation Resolution*: after shallow workflow correlation detection, in this part the focus resides on the deep analysis, and if possible resolution of any conflict. In the case that the running workflow conflicts with the incoming workflow, a disambiguation process is launched. This process decides whether it is necessary to preempt the running workflow, or rather the new workflow must be either eliminated or queued for later execution. In the case that the running process must be preempted, the systems global status is changed to preempt and then, when the Workflow processor checks for execution status realizes that the job must be stopped, hence, performing the required rollback operation (refer to the controller below for more details).

  Reads: Arriving workflow priorities and preemptiveness, running workflow status, priority and preemptiveness.
  Writes: Arriving workflow Next Status—Schedule, Pre-empt, or Cancel, and running workflow Next Status—pre-empt, or Cancel.

The workflow correlation considers aspects such as type of workflow, e.g., link provisioning, IP offloading, analyzes the semantics under the workflow and then detects any potential conflict between the new workflow and the workflows already enqueued and running on the system. In order to ease the correlation of workflows the smart analytics module will delay the execution of selected workflows with the objective of detecting possible collisions or duplicated workflows more effectively. Then, at this point the Smart Correlation resolution will perform the following action on the conflicting workflows:

- Conflicting workflow in running state:
  - Preempt the workflow if it is already running and the new workflow has more priority.
  - Cancel the running workflow and executing the new one.
- Conflicting workflow enqueued:
  - Reenqueue the existing workflow before the new one
  - Remove the workflow from the queues
  - Merge both workflows

The interface between the Management Controller and the Workflow Processor when initiating a workflow will be asynchronous, the reason for that is that a particular workflow can potentially require several minutes to execute, if synchronous this would cause timeouts on the protocol and, at the same time cause a management burden to the MC. Then, this initiation interface will be defined as follows:

- Input
  - TriggerTicket
  - AuthTicket
  - TriggerParams
  - Log
- Output
  - Status
  - log

Where the inputs determine the trigger and its parameters to be executed, jointly with logging information to the WFP. And the system will return OK in the case of successful workflow initiation, or ERROR with the proper status message in the case any issue was detected on the workflow.

Controller

This component has one main goal, i.e., to assist the Workflow processor in the task of determining whether the Job on the run queue can continue its execution or needs to be pre-empted. This is accomplished by the management of a global status flag which determines if it is safe to continue the execution of the particular workflow.

The internal workings of the three different blocks, namely, Workflow status update, Setup pre-emption strategy, and Status verification, have the task of managing the monitoring of the status progress of the workflow processor. To this end, as instructed by the Job correlation, when a particular flow must be pre-empted, the system will determine through the pre-emption strategy whether the flow will be paused or restarted.

This controller is a passive service that will be invoked periodically from the WFP, the objective of such invocation is twofold, primarily it is focused on informing the Management Controller about the execution status of the workflow, and second to open a communication mechanism to the BPEL internals allowing to cancel, or pre-empt the workflows. The interface prepared for this purpose is:

- Input
  - TriggerTicket
  - StatusCode
  - StatusMessage
  - Log
- Output
  - New Status

Hence, when receiving an update from the WFP, the MC will validate that the workflow can continue its execution, and return a new status accordingly. The possible options in this regard are: CONTINUE, CANCEL, and ROLLBACK.

To avoid possible starvation of the MC in case of WFP failure, if there is no Status message from the WFP during 3 minutes, the MC will assume that the WFP is down, and will inform the administrators the issue.

Finally, the only configurable aspect of the Smart Analytics concerning the Administration interface is the correlation window as detailed in Section 12.1.3, where it is possible to define the span of time a workflow will be delayed waiting for potential collisions with other incoming workflows.

4.1.3 Scheduler

The scheduler is the component within the *Management Controller* responsible of managing the Wait and Run queues. In the design of the *Management Controller* we took particular care in the fact that the queues had to be managed by a single entity, not allowing distributed write access. The rationale behind this idea is that the Scheduler is a critical component, since it is the part responsible of the actual execution and prioritization of the jobs present in the system. In order to perform its tasks, the Scheduler is composed by two different parts, the first one which has already been depicted in Figure 1. And the second one as shown in Figure 5 that is a standalone process monitoring the status of the queues, and when a new job appears it is in charge of executing it.

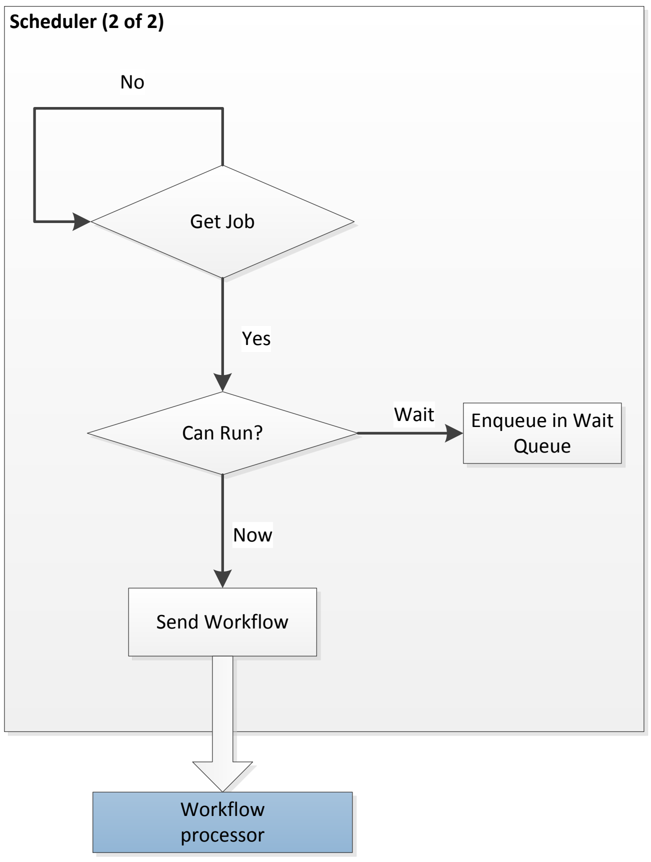


Figure 5. Second part of the Scheduler building blocks.

When a new job, incoming from a new trigger needs to be attended by the Scheduler, the following steps must be performed:

- *Workflow Adaptation* (see Figure 1) given a predefined set of policies, a new priority is assigned to the workflow, at this moment, it is also decided whether this workflow should pre-empt another running workflow.

  Reads: Job assigned priority and pre-emption
  Writes: Job priority and pre-emption

- *Workflow scheduling,* at this point the system must decide whether this job is ready to run or not. In the case it is (no correlation) the Job is inserted to the Wait Queue, from where it will be passed to the run queue by the second part of the scheduler.

  Reads: Job next status
  Writes: Job Next Status: Schedule, Decide pre-emption

- *Job queue* is the actual block where the job is inserted into the queue.

  Reads: Job
  Writes: To the corresponding Wait queue

- *Finish Job,* when a workflow finishes its execution, from the Controller a signal is sent to the scheduler in order to remove it from the queues

  Reads: Running workflow status
  Writes: Run Queue

From the administration point of view it is possible to configure two different aspects of the internal Scheduler, the length of the different queues and the queues able to pre-empt jobs already running on the system, the detailed description of these interfaces can be observed in Section 12.1.3

.

4.1.4 Logging and Monitoring

Logging is a fundamental requirement of any system, as we need a way to diagnose and to isolate the cause of a system malfunction. Information logging helps fixing problems, analysing system health, analysing problems, and preventing future malfunctions in the system by analysing the log information. The ONE adapter system log file contains events that are logged by the ONE core and auxiliary modules. The ONE adapter log files may contain information such as device changes, device drivers, system changes, events, operations, among other types of log information.

The logging and monitoring (LM) sub-module consists of a database that stores all logging information that has been sent by any module (i.e., both the core modules and the auxiliary modules). The stored data can then be accessed by an ONE administration functionality for statistic generation or for identifying any irregular behaviour of the ONE adapter and can issue notifications via E-Mail or SMS. It is possible that the ONE administration functionality is supported in this activity by a log-analyser, which automatically checks for certain events in the logs. Figure 6 shows how the LM module is connected to the other ONE adapter modules.

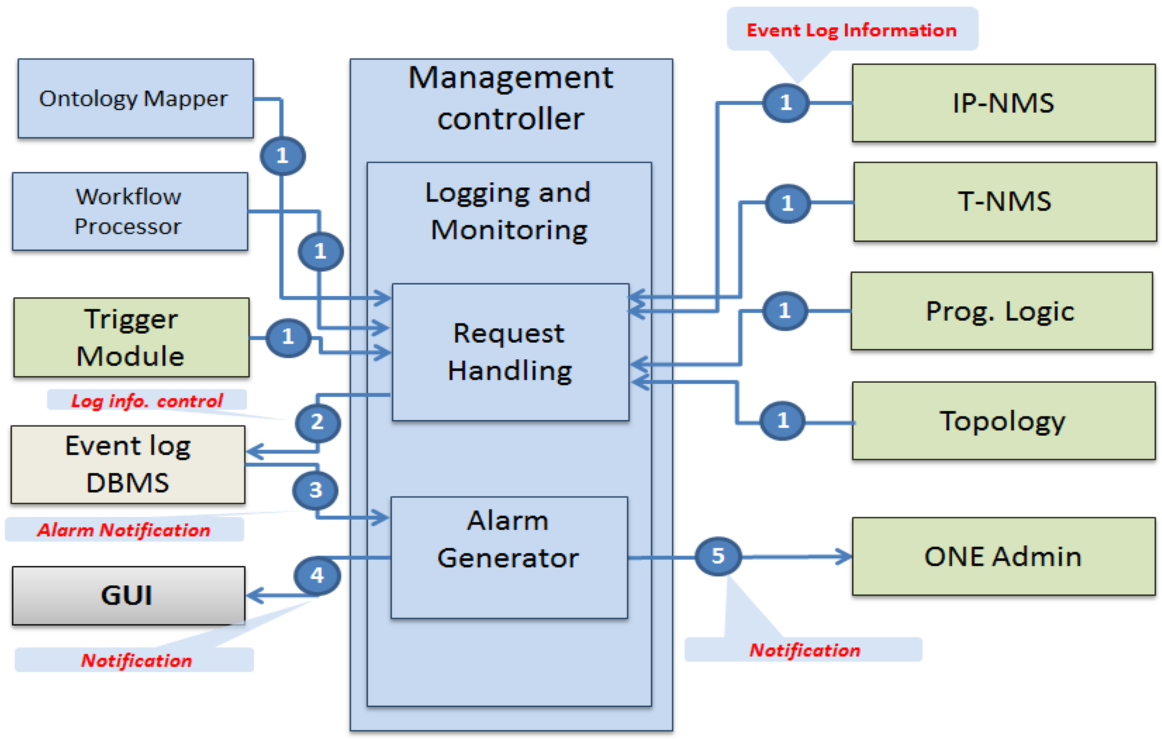


Figure 6. Logging and Monitoring functional modules and its interfaces.

As shown in Figure 6, the Logging and Monitoring module is connected to all core and auxiliary modules. The above mentioned interfaces allow every module to access the LM module so as to store its status into the Event Log DBMS. The storing is done by the Request Handling function. The LM module consists of another sub-module, namely the Alarm Generator module, which is designed to generate alarm notifications to the ONE administrator or to the GUI.

The implementation of the Logging Module within the Management Controller benefits from the usage of the ESB. Given that, when a new message is exchanged among the modules, such message is forwarded to the Logging Module, hence centralizing the logging facilities.

In coordination with the Notification Module, once the logging message is received it may be relayed to the GUI.

4.1.4.1 Notification Module-Proxy

The Notification module is a sub-module of the *Management Controller*. Its aim is to notify users of the ONE adapter via certain services such as email, SMS, etc., the status or activity of the ONE Box. By activity or status it is understood all the events that have been logged by the Logging and Monitoring module, such as the execution of workflows. The notification task could be performed in specific intervals of times (every hour, day) or on an event-basis, meaning, every time a certain event occurs in the ONE adapter, such as, the execution of a workflow.

The Notification module offers a very handy feature to network operators. In a network operator scenario, the ONE adapter is intended to be used by various final users. It is crucial from the point of view of a network operator to keep every user informed on the activities performed by others.

The interfaces of the Notification Module are shown in Figure 7. This module communicates via web services with the GUI to receive any administration request. Via this interface the notification module retrieves a notification object. The notification object is a JSON array containing the user's contact information (e.g. email, phone number, etc.), the interests of each user, these are the events a user is interested in, and time intervals in which a user would be interested in being notified.

The notification module has an interface to communicate with the Logging and Monitoring module. The purpose of this interface is to obtain the activities logged for the ONE adapter. These activities will be collected in a time window, specified for each user and defined by the ONE Admin module.

As mentioned before the interfaces of the Notification Module that communicate to the GUI and the Logging Monitoring Module are based on Web Services, on both of them a JSON string is expected as a result.

From the administration perspective, the notification module will allow the configuration of new trigger endpoints, which are the different actors receiving the log messages. The relevant interface definition can be seen in Section 12.1.4.

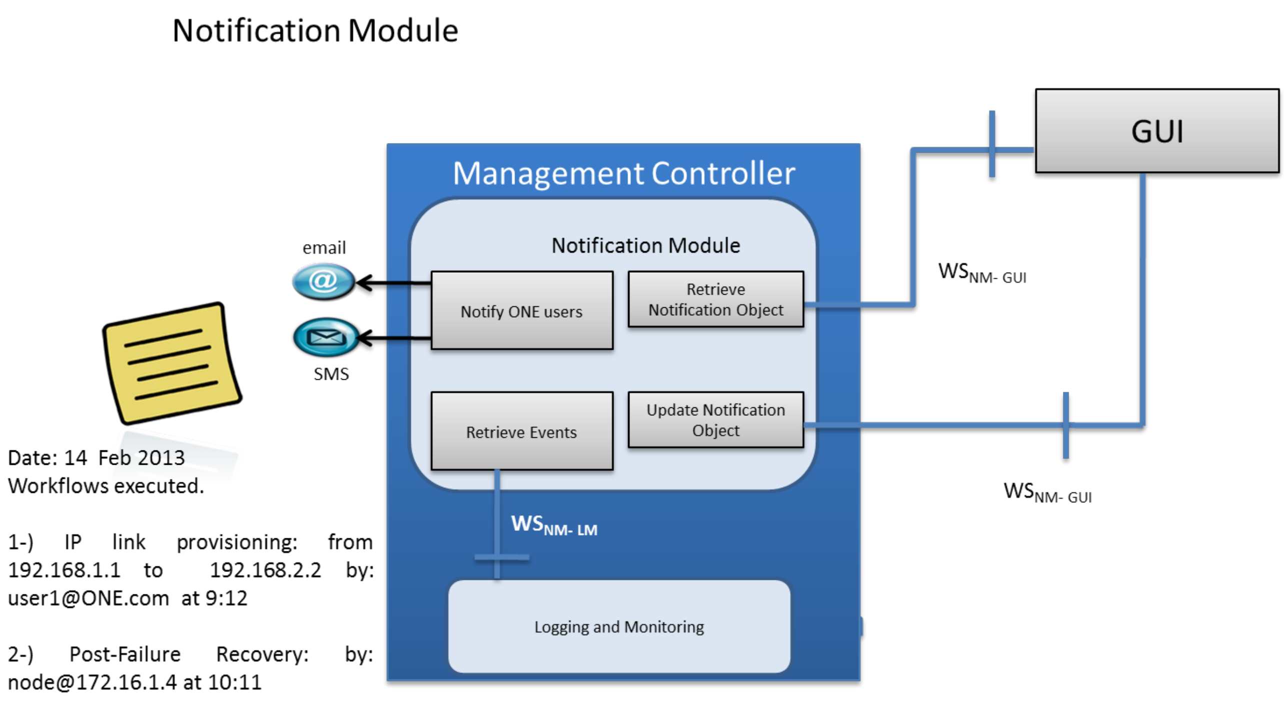

Notification Module
GUI
Management Controller
Notification Module
email
SMS
Notify ONE users
Retrieve Notification Object
WSNM- GUI
Retrieve Events
Update Notification Object
WSNM- GUI
WSNM- LM
Logging and Monitoring
Date: 14 Feb 2013
Workflows executed.
1-) IP link provisioning: from 192.168.1.1 to 192.168.2.2 by: user1@ONE.com at 9:12
2-) Post-Failure Recovery: by: node@172.16.1.4 at 10:11

Figure 7 Notification Module Interfaces.

4.1.5 Administration of ONE Services

The modular nature of the ONE adapter architecture implies that the configuration and administration of modules is not significantly dependent on each other. As a result, rather than delegating the administration of the ONE adapter to a single centralized administration module, each module in the ONE adapter exposes administration interfaces that allow users to configure the modules individually. To ensure uniformity, the modules however have to ensure that the user is authorized to perform administration functions with the AAA before performing any administration function as seen in Figure 8. Administrative privileges to users can be assigned per task or per module, and when making the request to the AAA, the module includes the Authentication Ticket and the Operation reference. The AAA verifies if the user is authorized to perform the specified administration operation and responds accordingly.

The use of module specific administration also eases, to an extent, the integration effort for services used to communicate with third-party systems, which do not need to be integrated with a centralized administration module, but only need to use the standard authorization interface provided by the AAA.

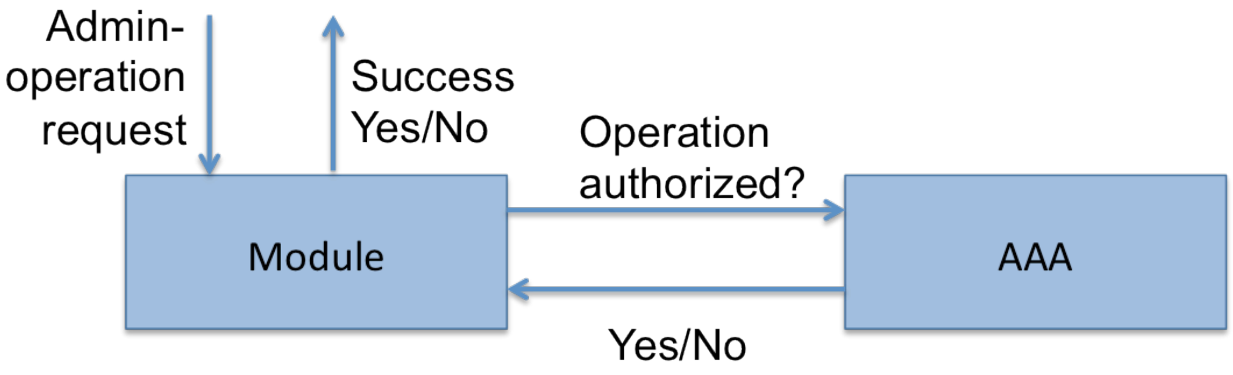


Figure 8 Request for an administration operation

While most modules are independent of each other, there may be some dependencies that need to be incorporated between the modules. For example, the workflow processor and the trigger module need to have a consistent trigger definition in order to map triggers to workflows within the ONE adapter. Therefore, the administration services of these modules must be designed to take into consideration any dependency with other modules in the ONE adapter ecosystem,

In this document, we will describe the administration functions associated with the modules along with their module description, but the authorization mechanism in all these functions is assumed to be the same as described above.

4.2 Ontology Mapper

The Ontology Mapper (*OM*) is one out of three of the core modules of the ONE architectural design. It is responsible for the internal ***syntactic adaptation*** of service requests supported by the ONE adapter for external communication. Its main functionality is the adaptation from generic representations to technology-specific forms for enabling communication with external actors.

Within the ONE adapter, inter-communications are intended to be agnostic of the external network infrastructure, meaning that, operations within a workflow are defined regardless of device manufacturers, Operating System (OS) versions, communication protocols, proprietary language-specifics, etc. However, the Ontology Mapper is the entity responsible of mapping from agnostic-forms of communication to specific technology-dependent forms. The OM provides the means for communicating with external entities. To this end, the OM is based on the use of pre-defined ontological definitions stored within the Ontology Repository. These pre-defined ontologies are vendor-specific and automatically generated based on ***semantic interpretations***.

In short, the Ontology Mapper provides a two-fold functionality. On the one hand, it ***semantically interprets*** the communication "language" (e.g. command set for multi-vendor CLI configuration, MTOSI, etc.) of external available devices to automatically generate specific-instances of ontologies bearing its semantic content, and on the other hand, based on these generated device-specific ontologies it provides ***syntactic adaptation*** of general agnostic requests within the ONE Adapter.

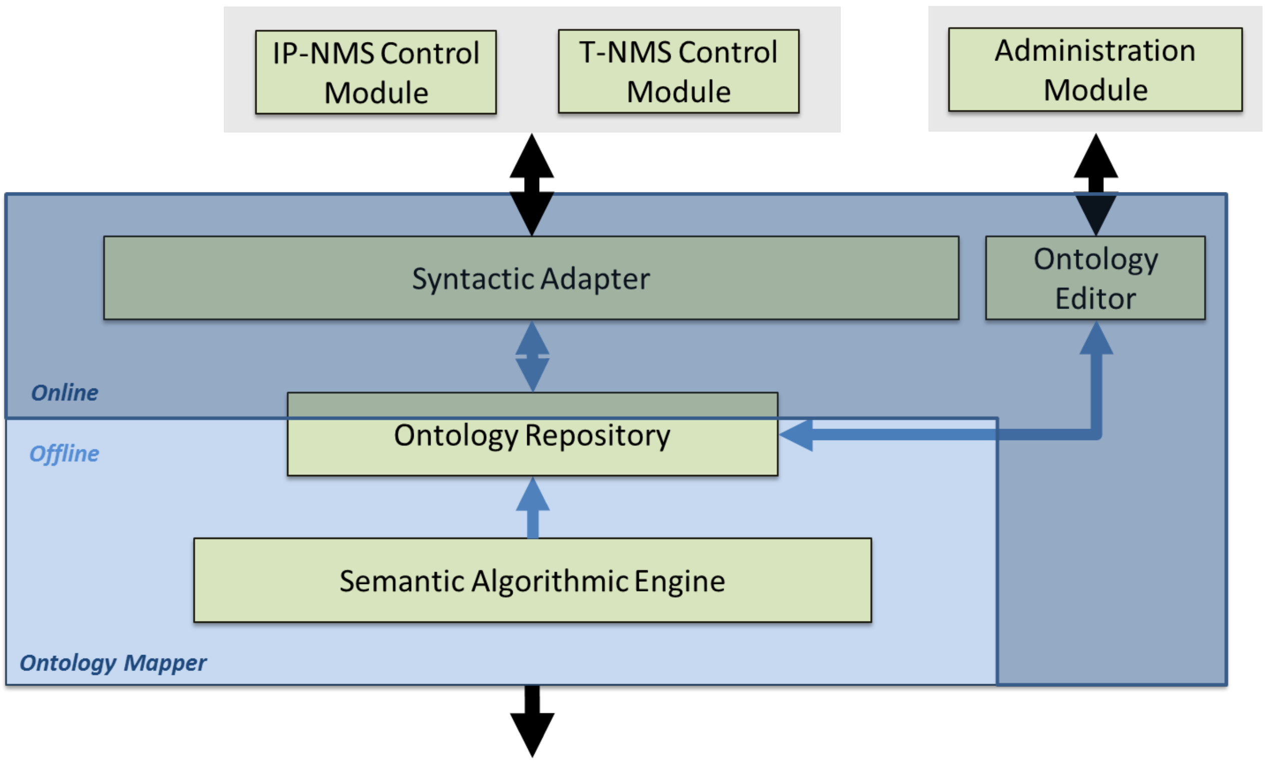


Figure 9 Internal Components of the Ontology Mapper.

Figure 9 depicts the internal representation of the Ontology Mapper; we can mainly distinguish among four components, namely: *Semantic Algorithmic Engine*, *Ontology Repository*, *Ontology Editor* and *Syntactic Adapter*. From the figure, we can notice that two modes of operation are differentiated: *offline,* which involves the *semantic algorithmic engine* and the *ontology repository,* and *online* which involves the *ontology repository*, *ontology editor* and *semantic algorithmic engine*.

The *offline* mode comprises all the operations that can be executed previous to the actual operational run of the ONE adapter. During this phase, technology-dependent ontological definitions are built within the *semantic algorithmic engine,* based on the use of decision algorithms, extraction methods, data mining tools, etc. These technology-dependent forms of knowledge will be stored within the *ontology repository* for further use during the operational run of the ONE Adapter. The *online* phase refers to the operations executed during run time, mainly related to the request of syntactic adaptations of the supported services for external communication (e.g. formal syntax for requesting the external T-NMS to set a light path between two end-points).

Semantic Algorithmic Engine

The *Semantic Algorithmic Engine* provides the mechanisms for generating specific instances of the general domain ontology. The general domain ontology refers to an agnostic representation of the domain's knowledge, to the highest level of abstraction. While a specific instance refers to a version of the same ontology which is linked to the specifics of a certain device or communication "language". A practical example is given for the case of router configuration. For the configuration of routers through the Command-Line Interface (CLI), the set of required commands may substantially differ from one vendor to another and even within a same manufacturer from model to model. For this reason, "specific" instances of the general routing ontology require to be generated on a per-vendor / per-model basis. The general ontology will comprise all the concepts within the routing domain regardless of vendors, models, IOS versions, etc. while instances of this ontology will be specific for each vendor (e.g. Juniper, Cisco, Huawei, etc.) or even specific to a router model, this depends on the level of divergence of the command set. The specific instances of the routing ontology will provide information on the set of commands for the configuration of the device. Later in this chapter, we'll provide further details on the inputs, processing and outcomes of this internal component.

Ontology Repository

The *Ontology Repository* is responsible for storing all specific ontology definitions. When syntactic adaptations are requested from an auxiliary module to the Ontology Mapper, ontology instances are looked up in the repository according to the external device specifics (e.g. router vendor, OS version and router model).

Ontology Editor

The *Ontology Editor* provides an interface to update, add, and delete ontology definitions. This functionality is essential to provide administrative control over this module from a user level guarantying the flexibility and evolution of the ONE adapter. It supports automatic updates of ontologies in the case that a router upgrade leads to a new release of an operating system or when starting utilizing a new measurement tool after system migration, etc. It can also support manual edition of ontologies for further low-level user interactions. All these administrative functions are requested through the interface described in Section 12.3.2. It is worth highlighting that the functional behavior of the Ontology Mapper will not depend on the "tool" selected for ontology edition (though it would clearly depend on the ontologies per se and the mapping mechanisms). In any case, the Ontology Editor will support basic operations, such as importing ontologies developed by an external tool, importing directly from an external repository, exporting ontologies, adding mapping algorithms, automatic update of ontologies, etc. for improved usability.

Syntactic Adapter

The *Syntactic Adapter* provides the internal interface to request language adaptations from the auxiliary modules (i.e. IP-NMS and T-NMS Control Module). Based on the services that are provided by the Ontology Mapper, the *Syntactic Adapter* will query the Ontology Repository for an instance corresponding to the specifics of the external device, once an instance has been found it will retrieve the specific syntax corresponding to the requested service. For example, following the same case of the router configuration, if a *set static route* service is requested for a Cisco 7200 Router, it will retrieve the set of commands compatible with this router model for external communication and configuration. The interface exchange for this example will be the following (see Appendix B – Ontology Mapper Module - IPNMS Configuration Request, for interface definition):

```
.setJSONInput({"routerVendor":"CISCO","routerModel":"CISCO7200","routerIOsVersion":"12.4","action":"Con
figureStaticRoute", "confMode":"CLI","optParams":{"destinationIP":"192.0.2.0",
"destinationMask":"255.255.255.0", "nextHop":"192.0.2.4","distance":"12"}})
```

The output for this request will be a vendor-specific statement for the configuration of a static route on a CISCO router:

```
{"module":"OntologyMapper", "sentence":"enable \n configure terminal \n route 192.0.2.0 255.255.255.0
192.0.2.4 12}
```

In the scope of the ONE Adapter, the Ontology Mapper will provide services to the IP-NMS and to the T-NMS Control Modules, mainly for configuration purposes at the IP and Optical layers, respectively. As introduced previously, on the IP Layer, remote device configuration is currently based on the use of the Command-Line Interface. The main issue behind this common practice is that due to the proprietary nature of this configuration mechanism, Network Managers require specialized knowledge for each type of existing device, adding to this, the fact that within a single vendor the command set may vary from model to model. For this reason, the OM will provide the means for abstracting this specialized knowledge from network managers, achieving truly automated mappings given the properties of the device to be configured. For the Optical Layer, configuration adaptations can be done through the T-NMS control module for MTOSI adaptations.

It is important to highlight that due to the heterogeneity of underlying configuration protocols for IP-based networks (a real practical problem for current operators with the use of CLI mechanisms as the preferred approach for network configuration), we will focus our implementation efforts on abstracting these complexities for the IP layer network configuration scenario. Next, we'll provide details on our approach for achieving true automated configurations at the IP Layer based on the use of Ontologies. For more details on the definition of the interfaces of the Ontology Mapper module please refer to Appendix B: Ontology Mapper Module (Section 12.3).

From the Command Help to Automated Forms of Configuration for IP Routers

Figure 10 depicts the internal structure of the ONE Adapter with specifics for the case of interaction with the IP-NMS Control Module. The differences that will exist between this case and the case where OM interacts with the T-NMS Control Module is basically related to the Offline phase, meaning that, if different instances of a general ontology are required, the algorithmic engine can differ according to the preferred approach.

For the IP Router Configuration case, the *Semantic Algorithmic Engine* will generate specific instances of the routing domain ontology based on the shared semantics between the help of the command set available for each router and the general knowledge represented by the meta-ontology. The final goal is to allocate IP router commands in an ontological definition containing all the semantics concerning the domain (referred in Figure 10 as: *general routing domain ontology*). The semantics, descriptions and information provided by the helps of the routers will provide the inputs (step 1) for a decision algorithm to accurately allocate commands within the concepts of the general routing ontology (step 2), as a result, we will store in the repository (step 3) specific instances for each configuration "flavor" (per vendor, IOS version, router model, etc.). During the online phase (steps 4 to 8), whenever a syntactic adaptation is requested (step 4), the syntactic adapter will query the ontology repository (steps 5 and 6) for the ontological definition supporting the external device and retrieve the command or set of commands corresponding to the requested service (steps 7 and 8). Figure 11 provides a detailed flowchart for the *syntactic adapter* concretely adjusted to the IP Router Configuration case. However, this flow of actions generically extends to the T-NMS Control Module interaction scenario. The Ontology Mapper activates with the request of a service either from the IP-NMS Control Module or the T-NMS Control. The Ontology Mapper has basically two means for ending its operation: either successfully retrieving the syntax for external communication or generating errors due to: (a) Unsupported model / communication protocol, meaning that, within the Ontology Repository no instance of a valid ontology is found (b) The service is not supported by the requested external device (e.g. a certain configuration service is not featured/offered by a given router model) or (c) Given the semantics within the ontological definition there are missing configuration parameters.

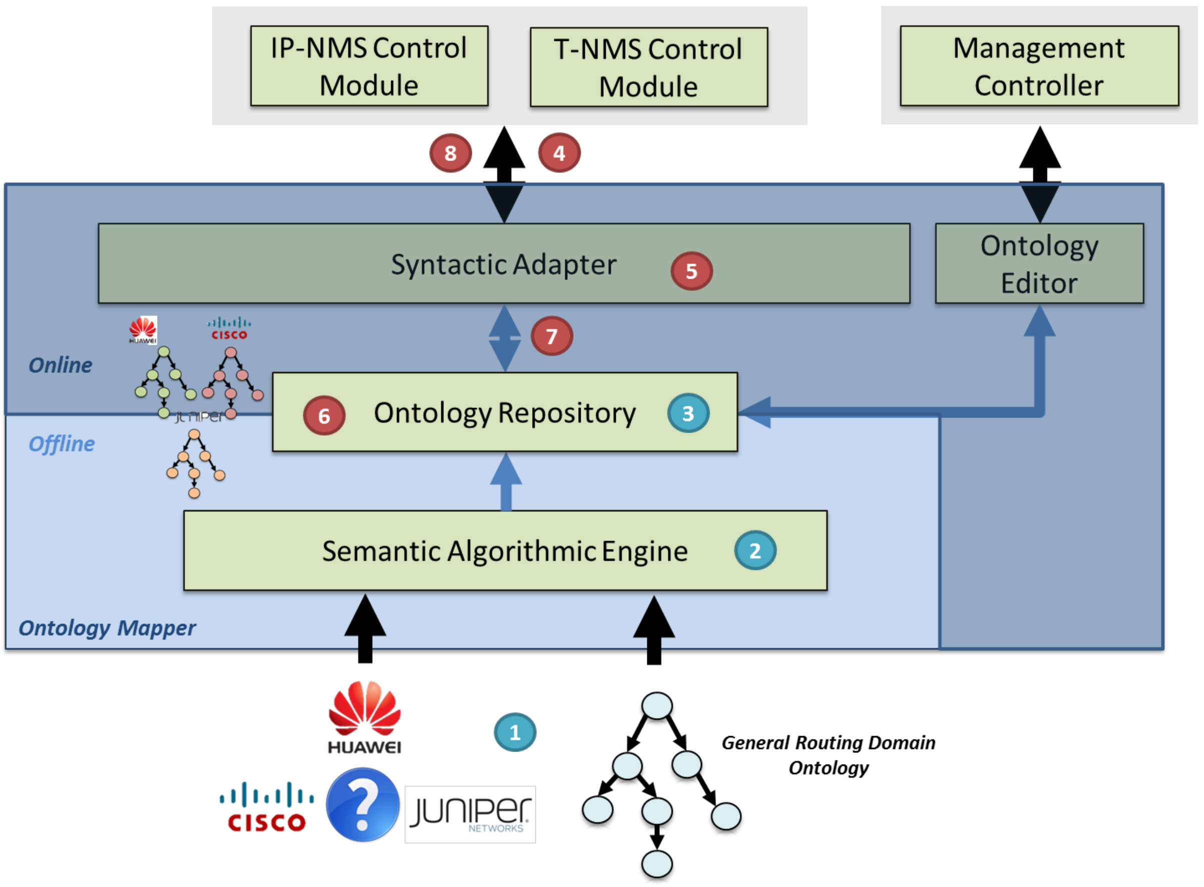
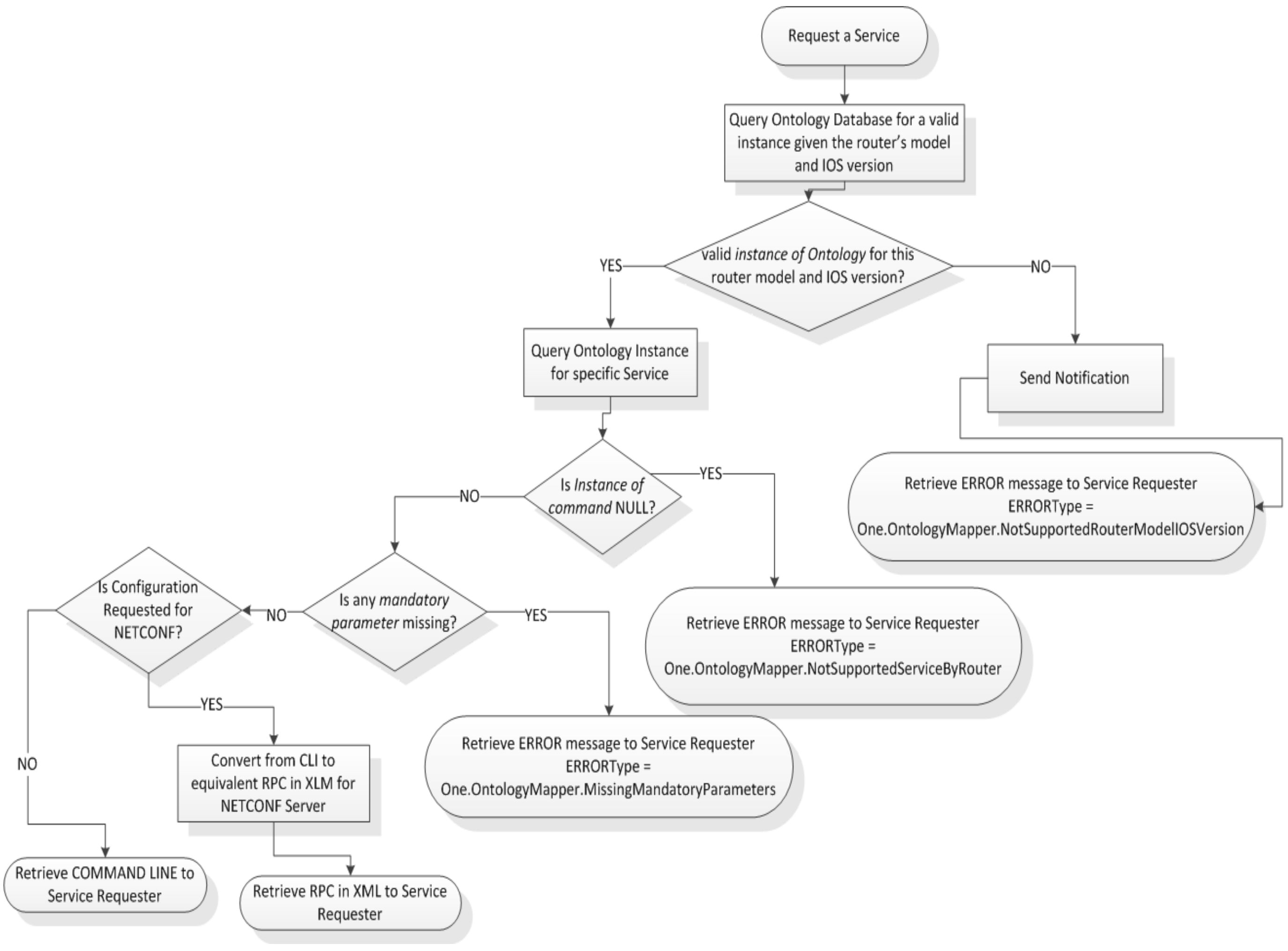


Figure 10 Internal Components of the Ontology Mapper – An Example for IP Router Configuration

Configurable Helps

As explained previously, our approach pushes toward the generation of specific instances of an ontology, based on the use of the routers command set HELP. Taking the semantics of the routing domain from a general “un-flavored” ontology, we will enable the allocation of commands through specialized software agents capable of controlling, managing and reasoning over the information and basic semantics provided in the <HELPS>.



Request a Service
Query Ontology Database for a valid instance given the router's model and IOS version
valid instance of Ontology for this router model and IOS version?
YES
NO
Query Ontology Instance for specific Service
Send Notification
Retrieve ERROR message to Service Requester ERRORType = One.OntologyMapper.NotSupportedRouterModelIOSVersion
Is Instance of command NULL?
YES
NO
Retrieve ERROR message to Service Requester ERRORType = One.OntologyMapper.NotSupportedServiceByRouter
Is any mandatory parameter missing?
YES
NO
Retrieve ERROR message to Service Requester ERRORType = One.OntologyMapper.MissingMandatoryParameters
Is Configuration Requested for NETCONF?
YES
NO
Convert from CLI to equivalent RPC in XLM for NETCONF Server
Retrieve COMMAND LINE to Service Requester
Retrieve RPC in XML to Service Requester

Figure 11 Ontology Mapper – Syntactic Adapter Workflow for IP Router Configuration Example

Aimed to proof that the decision's algorithm performance may improve as more verbose and enriched help descriptions are provided by the command help set, in the context of OPENER [OPENER12] we've developed a prototype implementation which enables adding, removing and editing help features at the time that we can append meta-information to available commands. Our final goal is to demonstrate that more verbose helps can not only provide manual users with richer semantics but at the time it can ease the task of automated tools and software agents for semantically distinguishing among commands.

4.3 Workflow Processor Module

The Workflow Processing module (WFP) is responsible for the management and execution of workflows in the ONE adapter. The choice of web services, as especially BPEL [BPEL01] to define workflows in our implementation has forced architectural constraints on the WFP that are reflected in the architecture presented in Figure 2.

The WFP consists of three different functions, namely the Workflow Database, the Workflow Initiation function and the Workflow Execution Status function. We now describe each of them in detail.

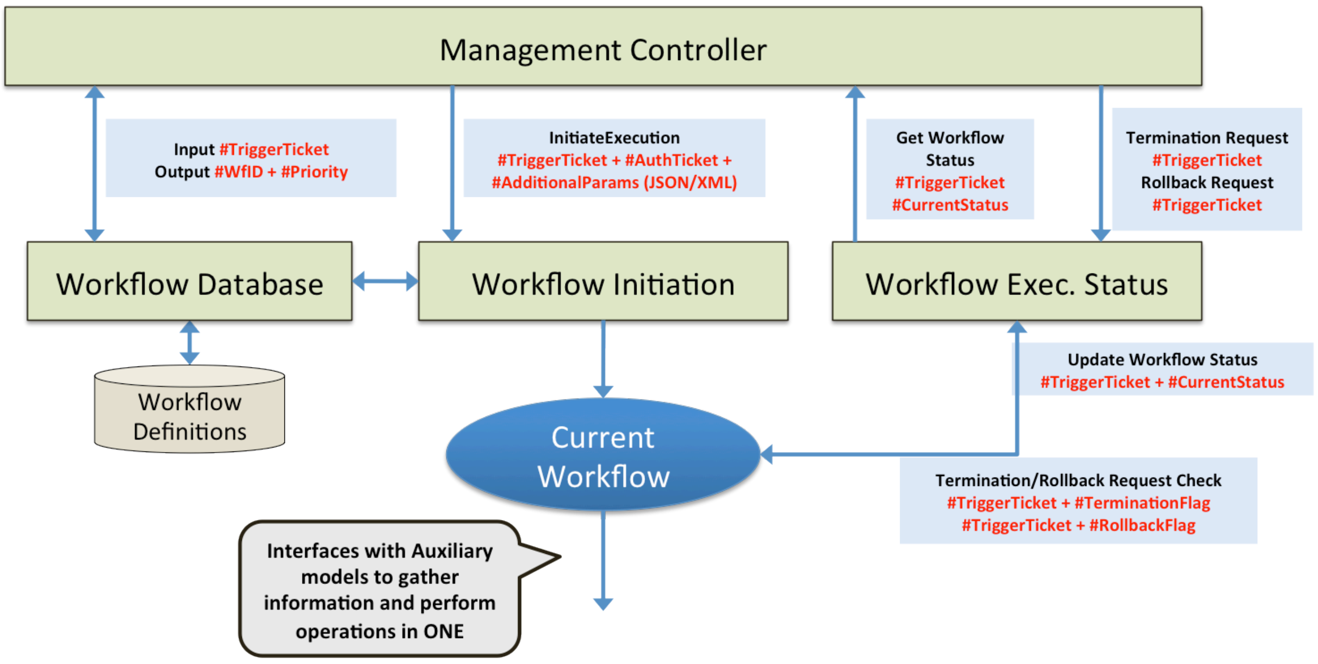


Figure 12.Components and their Interaction in the Workflow Processing Module.

Workflow Database

The workflow database provides a set of interfaces to the management controller to get information about the workflows available in the WFP required for smart analytics. The interface specification in the Appendix (cf. Section 12.4.1) indicates that each workflow is associated with a specific *triggerID* and a static *priority* which is used for scheduling workflows in the Management Controller. The management controller uses the *triggerTicket* to request information about the associated workflow and the priority from the workflow database in this scenario.

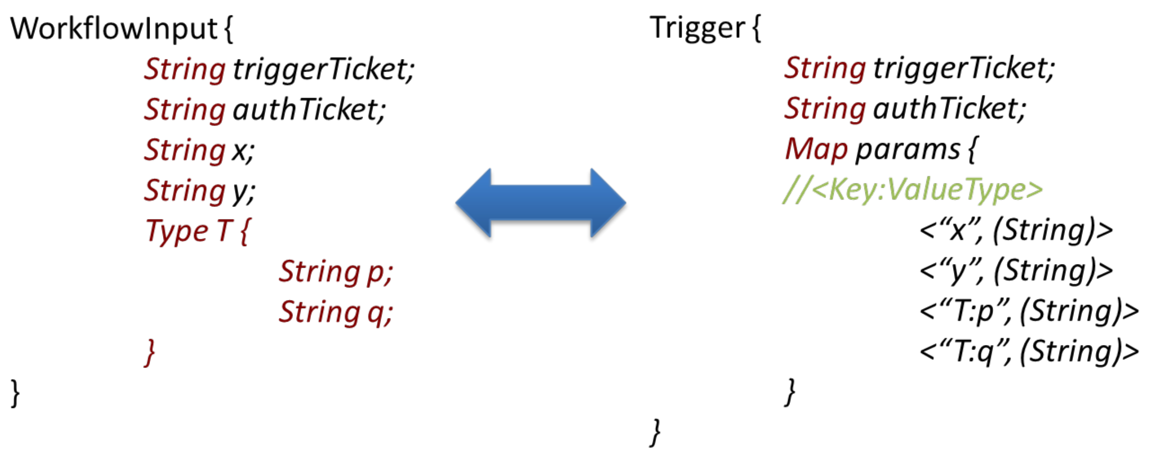
Workflow Database Administration

The ONE adapter allows for dynamic insertion of workflows during run-time, and each workflow is associated with an interface specification and an end-point location (i.e. the location where the workflow is deployed). The administration interface in the workflow database module (cf. Section 12.4.2) allows users to add/delete/modify the interface specification, the end-point location and the description for a workflow within the ONE adapter. The interface also allows users to add/delete/modify workflow associations with trigger IDs and priorities for scheduling execution of triggers within the ONE adapter.

Workflow Initiation

The workflow initiation function provides an interface to the management controller to initiate a workflow execution in the ONE adapter. It receives as inputs the Trigger as generated in the Trigger Module. As each trigger is uniquely mapped to a workflow, it uses the *triggerTicket* parameter to get the corresponding workflow definition from the workflow database. In our model, due to the choice of BPEL, workflows are implemented as stateless services. In this scenario, the Workflow initiation, upon receiving the workflow definition, initiates execution of the workflow (which is in effect a service). As the workflow execution times may be long, all workflows are designed to be processes which only provide the status of their execution by the workflow Execution status function (defined in the next section) and do not return any parameter at the end of the execution. Therefore, the workflow initiation function does not return any parameter upon completion of execution of a workflow.

The interface for workflow initiation is presented in Appendix B (cf. Section 12.4.3) and includes the trigger ticket and the authentication ticket as mandatory parameters, which are associated with all triggers, while other parameters are included in the form of a string based Map data structure such as JSON. This way, the inclusion of a new trigger or workflow does not affect the interface specification between the management controller and the workflow initiation function. Note that there may be some pre-processing required to map internal trigger definitions (received as inputs by workflow initiation) to the input parameter specification of a workflow. In our architecture, to ensure that workflows can be included in run-time, a fixed naming convention is adopted, where parameter names in the internal definition match the input parameter names of a workflow. Another challenge is seen in case input parameters to a workflow have a nested structure definition: an example of the same can be seen in Figure 13. Here, the naming convention for the keys in the flat Map structure is fixed to reflect the hierarchical parameter specification required as workflow inputs. Note that this is the reason for carefully defining the Trigger Definitions in the Trigger Module.



```
WorkflowInput {
        String triggerTicket;
        String authTicket;
        String x;
        String y;
        Type T {
                String p;
                String q;
        }
}
```

```
Trigger {
        String triggerTicket;
        String authTicket;
        Map params {
        //<Key:ValueType>
                <"x", (String)>
                <"y", (String)>
                <"T:p", (String)>
                <"T:q", (String)>
        }
}
```

Figure 13. Mapping between Workflow Inputs and Internal Trigger Definitions.

Workflow Execution Status Function

Given that in our architecture, workflows are stateless operations, it is challenging to communicate in real time with an executing workflow process in order to issue termination/rollback requests. To facilitate the same, we use the Workflow Execution Status Function, which is used to store state of the workflow execution externally to the workflow itself. Every workflow in the ONE adapter is designed in a generic fashion that includes two internal operations after every generic operation involving invocation of an operation in an auxiliary module. These can be seen in Figure 14. After any operation that involves initiation of an operation in an auxiliary module, the workflow first updates the state in the workflow execution status function. During this period, if a termination or rollback operation is invoked by the management controller, this state is stored in the workflow execution status function. The workflow then checks if termination or rollback is requested from the workflow execution status function and if so, terminates the workflow or starts rollback operations, based on request.

The interface to update the state of the workflow is defined in the Appendix B (cf. Section 12.4.4) and includes triggerTicket parameter to identify the workflow in question, a status code to define the class of status update (e.g. execution complete) while the status message parameter provides human readable details of the exact status of operation within the workflow. The status messages also act as a keep-alive function: as the workflow execution is an independent process, an unexpected termination of the same may not be seen by the management controller, which needs to incorporate timeouts for such a scenario. The arrival of a status message from a running workflow can be used by the management controller to ensure that a workflow is still operational while the lack of the same can be interpreted as an unexpected termination of the workflow at which point the management controller can either diagnose the problem or start a new workflow execution.

Note here that we assume that the rollback of a workflow is programmed into the workflow itself, and would be different at different states in the workflow. This is used as it may not be possible to program rollback operations in all different auxiliary modules in the ONE adapter based on their functionality. As a result, it may also be required that the workflow needs to store some internal information during execution to facilitate the same. Therefore, if no rollback/termination operation is requested, the workflow updates the information required for rollback in future steps and then continues with execution.

The proposed constraints of workflow can hamper execution times, and therefore checks on rollback can be incorporated intelligently in the workflow, based on the estimated time required for operations to optimize execution times.

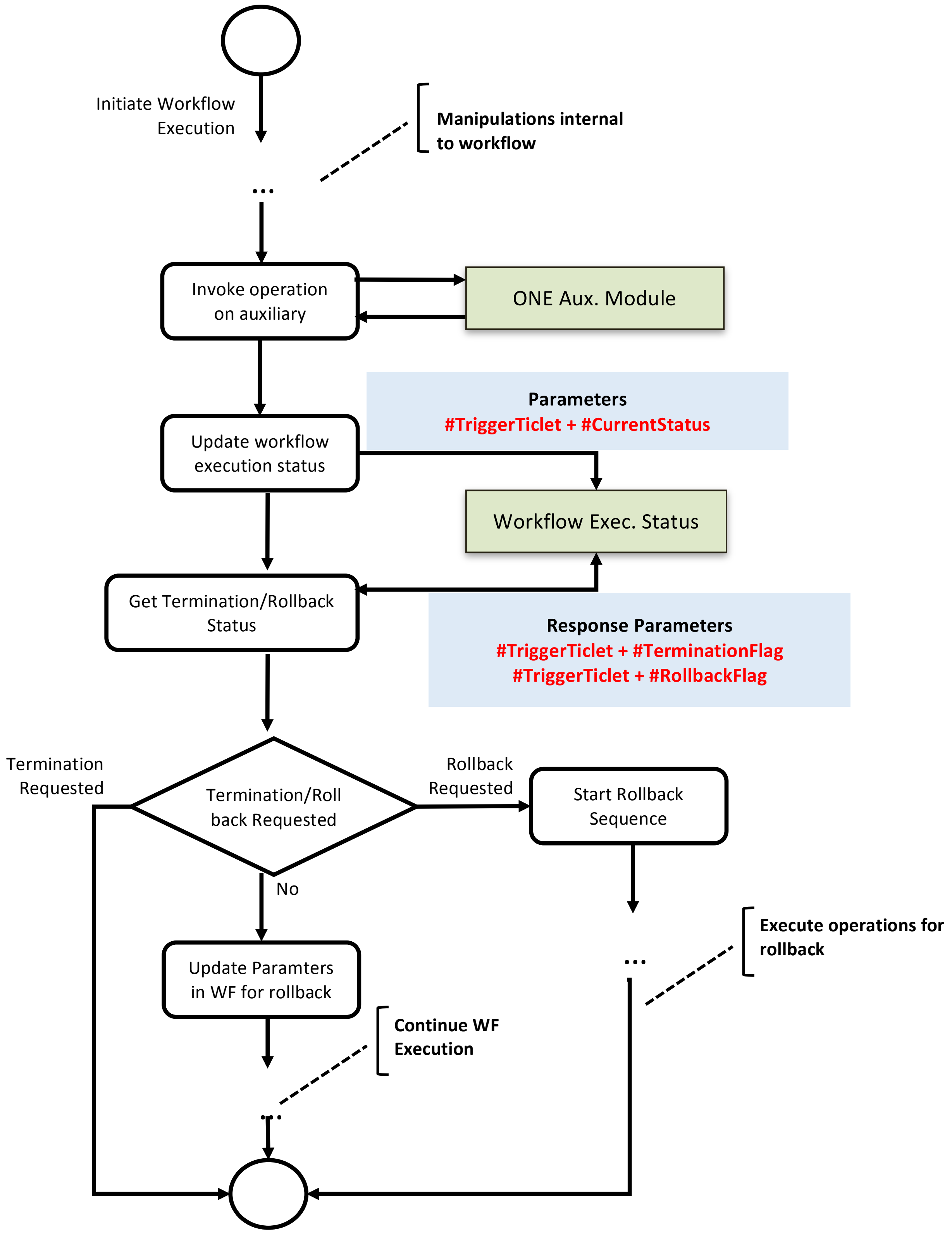

Initiate Workflow Execution
Manipulations internal to workflow
...
Invoke operation on auxiliary
ONE Aux. Module
Parameters
#TriggerTiclet + #CurrentStatus
Update workflow execution status
Workflow Exec. Status
Get Termination/Rollback Status
Response Parameters
#TriggerTiclet + #TerminationFlag
#TriggerTiclet + #RollbackFlag
Termination Requested
Termination/Rollback Requested
Rollback Requested
Start Rollback Sequence
No
Execute operations for rollback
...
Update Paramters in WF for rollback
Continue WF Execution
...

Figure 14. Generic Workflow for Updating Status and Initiating Rollback Termination.

4.4 GUI

The Graphical User Interface (GUI) is the interface between a human operator and the network equipment. The GUI offers an easy to use handling experience, while hiding the complexity from the user.

In Figure 15, the GUI is shown and its two components: The GUI Server and The GUI client.

The GUI Client is the front end and directly used by the human operator.

The GUI Server is the background functionality, used by the GUI Client and offers the following functions.

- Connection to the other components of the ONE adapter
- Repository for icons and background images
- Graphical Network Topology representation based on the information of the Topology Module
- Functionalities for Authentication, Authorization and Accounting for both human operator and modules of the ONE adapter by using the internal AAA ONE module
- Permission management for human end users and modules via the AAA module

The GUI will connect to the ONE adapter using a TCP connection. This connection must be always authenticated and authorized by the ONE AAA module by using encrypted passwords.

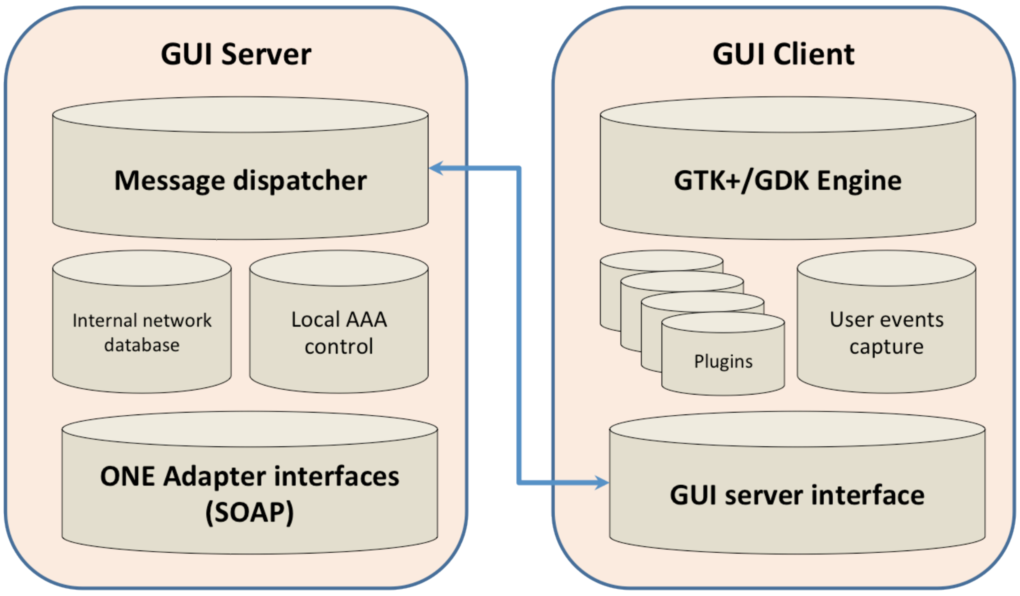


Figure 15. GUI structure

The available modules are:

- module_one: This module interfaces with:

    - Trigger module
    - Topology module
- module_one_logs: Used to get data from the ONE measurement module

The GUI Client is the human interface in the ONE adapter. It is composed by a graphical framework (Xges2) and a set of plugins. By default Xges implements a message processor, graphical interpretation of graphs and the capabilities to draw and edit it, the edition capabilities include the allocation of the nodes and setup properties for nodes and links, these properties can be sent to the GUI server and it will refresh the topology on all running instances Xges and all its modules, and as a consequence, modify the topology stored in the topology module database.

The plugins are dynamic libraries for Xges and have the three functions:
- Add low level graphical capabilities to Xges
- Expand the syntax and new type of messages to send/receive by the server side
- Expand the human user available operations, like new widgets to do same operations like manual link provisioning.

4.4.1 How the GUI server works

When the GUI Server starts, it will listen to a dedicated TCP port, while waiting for clients to connect. At this point the modules must be started and connected to the server. When a client, (module of Xges) connects to it, the server sends a public SSH key for user and password encryption. When the encrypted user and password is received, the GUI server deciphers it and calls the AAA module of the ONE adapter.

The AAA returns a response that indicates if the user is. If the authentication succeeded, the network view of the operator, admin level and session ticket parameters are returned. The permission level of the user is also set up and with this the permission to request operation and/or access functionalities within the ONE adapter

After this point, when at lat ONE client is connected, the server starts to dispatch messages from one client to another, (modules or Xges instances), serving and storing data like graphs, icons and configuration files in its internal database.

4.4.2 How the GUI client (Xges) works

Figure **16** shows the steps performed. In the beginning, the client loads the provided plugins to work; each plugin has three functions:
- Initialization: Here the plugin adds new widgets to Xges framework with the correspondent GTK+ events.
- Timer: Internal timed scheduled routines, e.g. flashing, showing and hiding of links

- Message processor: Intercepts the messages received by Xges and decides if operations are necessary, for example message modifications, calling Xges API functions or raising warning as a popup.

After the initialization, the GUI starts the login procedure requesting the username and the password. Then Xges opens a connection to the GUI Server side (1). The GUI Server then generates the private and the public SSH keys. The private key is stored inside the server memory while the public key is sent to the client side (2). The client receives the public key, encrypts the username and password and returns the encrypted authentication information back to the server (5).

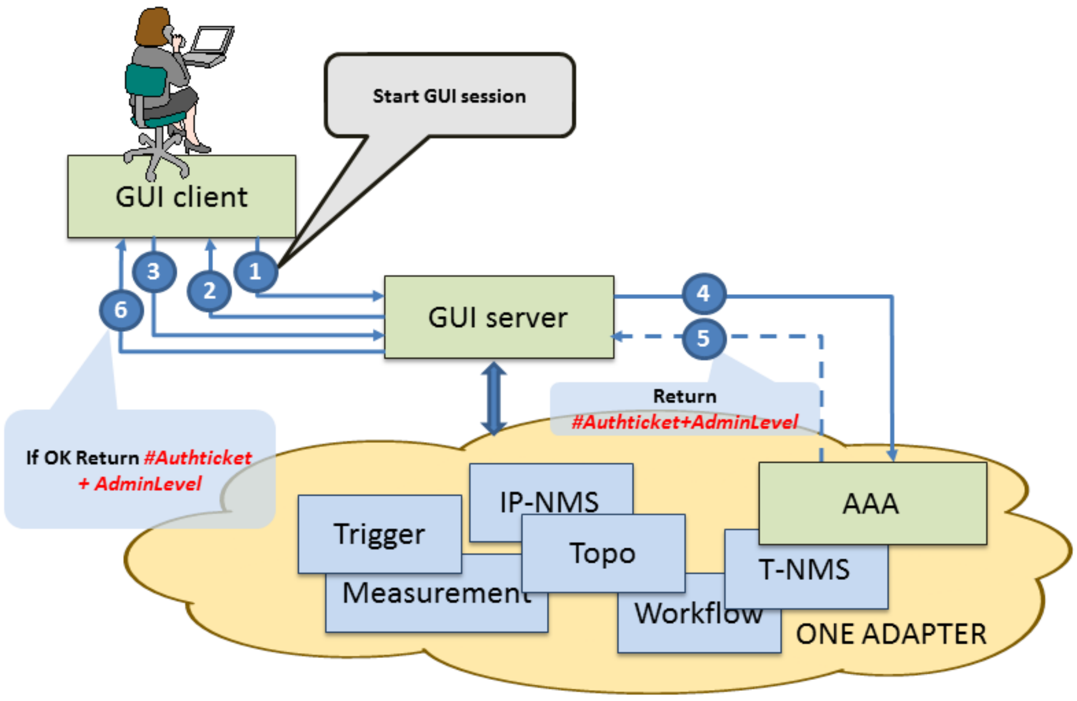
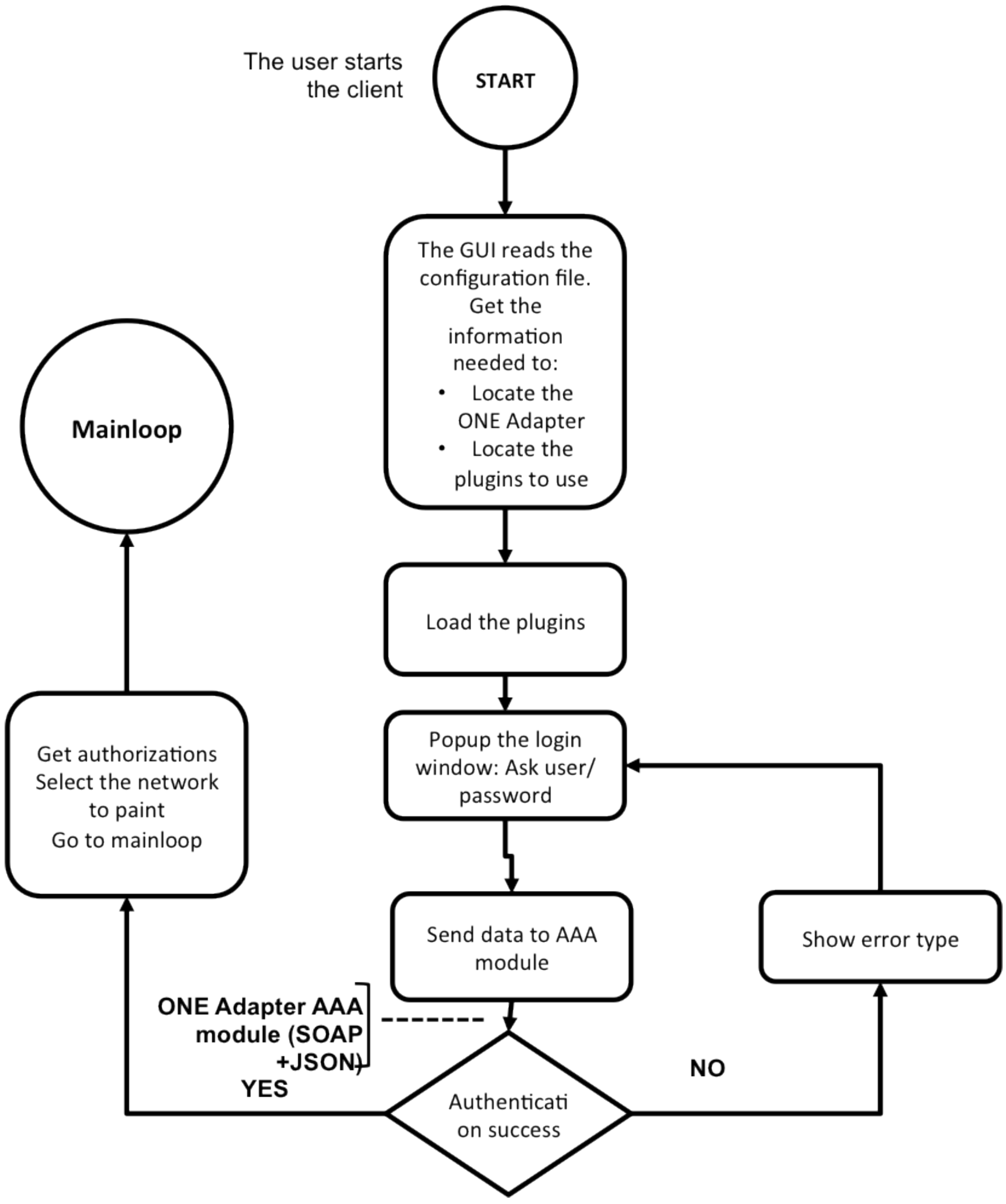


Figure 16 Login Procedure

The GUI Server will decipher the username and password, using its private key and send a request the AAA module (4) awaiting the response with the user permissions. The interface specification of the message can be seen in Section 12.12.1.
When the response is received (5), the GUI Server analyzed the permission. In case of an AAA reject, the GUI server sends a reject message (6) to the client and raises a popup warning and the User and password request starts again.

If the response has been positive, the server composes a session accept message, with the AndminLevel and AuthTicket parameters and it is delivered to Xges (6). The client stores the

AuthTicket and uses the Admin level parameter to show an appropriate look and feel behavior. Note that in any case, independently of the behavior of Xges, the server side of the GUI will ignore not authorized requests resulting in violation of Admin level parameter constraints, because the grants are managed at server prior to send the session accept message to Xges.



The user starts the client
START
The GUI reads the configuration file. Get the information needed to:
• Locate the ONE Adapter
• Locate the plugins to use
Load the plugins
Popup the login window: Ask user/ password
Send data to AAA module
ONE Adapter AAA module (SOAP +JSON)
Authenticati on success
YES
NO
Show error type
Get authorizations Select the network to paint Go to mainloop
Mainloop

Figure 17. GUI startup: Xges flowchart.

GUI Administration

The only administrative event for GUI is issued by the topology module. The topology module can ask to the GUI to refresh the current network view

In Section 12.12.5 the interface specification of the event administration is shown. It only requires the AuthTicket, with it the GUI correlates the network view with a topology. When the event arrives to the GUI, the GUI retrieves the new topology from the topology module.

4.4.3 GUI IP Link provisioning interaction example

This is an example of interaction between the GUI and ONE adapter in the IP link provisioning use-case.

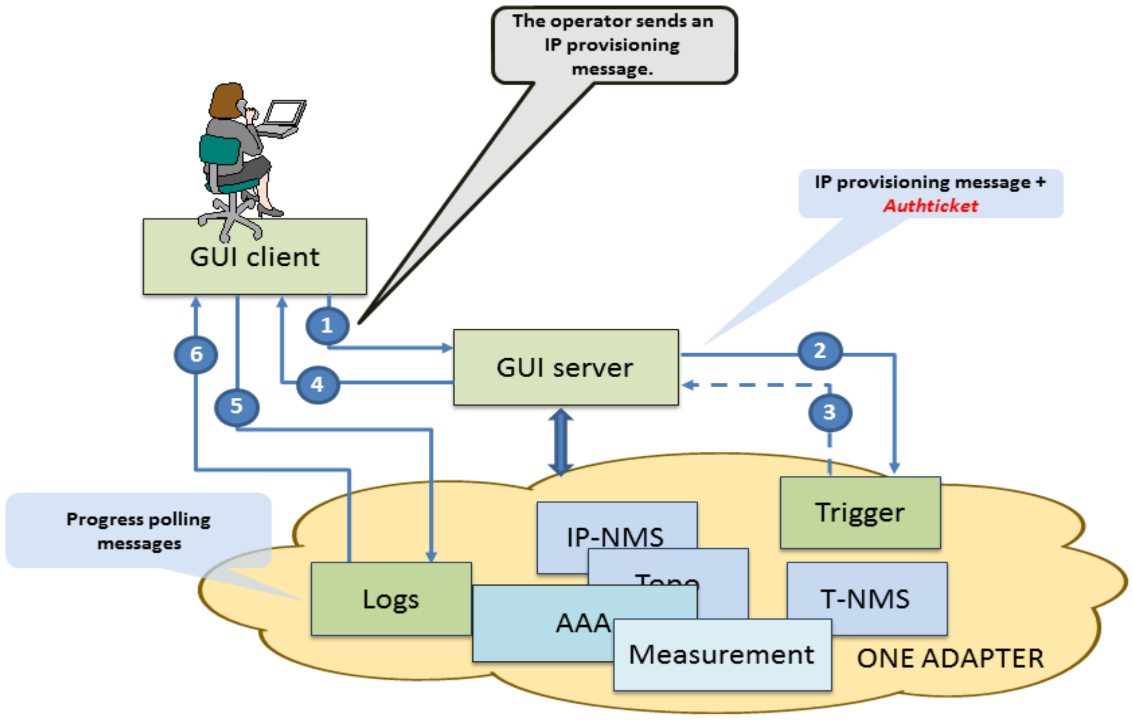

The operator sends an IP provisioning message.
IP provisioning message + Authticket
GUI client
GUI server
1
2
3
4
5
6
Progress polling messages
Logs
IP-NMS
Topo
AAA
Measurement
Trigger
T-NMS
ONE ADAPTER

Figure 18 IP link provisioning between the GUI and the ON adapter

(1) The operator defines and launches an IP provisioning event. The GUI client sends to the server side the endpoints, path and routing parameters using and internal data format.
(2) The GUI server translates the message to JSON and encapsulates it in a SOAP message with the trigger ticket and the Session ID. After that invokes the Trigger module WS with an http://one.eu/GUIEventListener/ReceiveGUIEvent message containing:

a. <?xml version='1.0' encoding='UTF-8'?><soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"><soapenv:Body><ns1:ReceiveGUIEvent xmlns:ns1="http://one.eu/GUIEventListener/"><EventID>GUI_PROVISION_IP_LINK</EventID><AuthTicket>3080082801</AuthTicket><Params>{"SourceRouterID":"MX240-3","DestRouterID":"MX240-1","SourceIpv4Addr":"130.130.130.2","DestIpv4Addr":"130.130.130.1","subnet":"255.255.255.252","SourceIfID":"ge-2/1/9","SourceIfVirtualID":0,"DestIfID":"ge-2/1/8","DestIfVirtualID":0}&#xd;</Params><log>{"logLevel":"20000","moduleName":"GUI","logMessage":"GUI {\"SourceRouterID\":\"MX240-3\",\"DestRouterID\":\"MX240-1\",\"SourceIpv4Addr\":\"130.130.130.2\",\"DestIpv4Addr\":\"130.130.130.1\",\"subnet\":\"255.255.255.252\",\"SourceIfID\":\"ge-2/1/9\",\"SourceIfVirtualID\":0,\"DestIfID\":\"ge-2/1/8\",\"DestIfVirtualID\":0}\r"}</log></ns1:ReceiveGUIEvent></soapenv:Body></soapenv:Envelope>

b. In this response can you see the following date coming from an actual capture:
    i. EventID: GUI_PROVISION_IPLINK
    ii. AuthTicket: 3080082801
    iii. The JSON string bearing the necessary data to provisionin the new link.

(3) The trigger ticket acknowledges the query responding:

a. <?xml version='1.0' encoding='UTF-8'?><soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"><soapenv:Body><ns2:ReceiveGUIEventResponse xmlns:ns2="http://one.eu/GUIEventListener/"><GUIEventReceived>true</GUIEventReceived></ns2:ReceiveGUIEventResponse></soapenv:Body></soapenv:Envelope>
Saying that GUIEventReceived is OK, (Simply received, not processed)

The response (3) is received by the GUI server side, translated to its internal protocol and delivered to the GUI client side. The client starts to poll the log module using its WS interface and providing the trigger ticket and the Session Id, sending regular messages messages (5) and (6), until an operation status finished is received

5 Functional Design of the Basic Auxiliary Modules

5.1 Trigger Module

The trigger module is responsible for accepting event notifications and using them to initiate workflows in the ONE adapter. The architecture of the Trigger module is presented in Figure 19. As described in Section 3, the trigger module contains one or more event listener's that receive event notifications both from internal and external entities. The external ones are users or devices in the network.

Each event listener is responsible for two operations: authenticating incoming events and generating a standard event description. All events have the same generic structure within the Trigger module as shown in Figure 20(A). Each event has a unique *EventID* and can have one or more parameters. The *AuthTicket*isis an authentication token associated with an event: for authentication, the event listeners communicate with the AAA module and provide device or user credentials, and the response from the AAA consists of a flag indicating if authentication was successful, and the *authTicket* which is used to re-confirm the authentication of the event at a later time with the AAA.

An event can have one or more additional parameters, which can change for different events. The definition of the Event is stored in the Event Definition database, which contains the *EventID* and a list of parameters associated with a specific event. Parameters in the Event Definitions can be specified as mandatory or optional, while data types of the parameters are defined either using primitives (*String*, *Integer*, *Boolean*, *Double*) or using Internal Data Objects, whose definitions are defined in Internal Data Objects. Internal Data Objects can either be a group of primitive or other data objects, similar to data structures in programming languages, and this definition is stored within the Internal Data Objects database. An example is the *IPSubnet* data type, which consists of an IP address and a subnet object. The Internal Data Objects can also be data objects that enforce specific formats or restrictions on primitive data objects: for example, the *IPAddress* data type is a *String* with a pattern-based restriction to check if the input string is actually a valid IP address. Apart from primitive restrictions, the internal data objects can also perform transformations. For example, in the case of a subnet, an incoming subnets value (which is a String) of the form "255.255.0.0" and "16" can both be converted to one format which is then used consistently inside the Trigger Module.

The additional parameters are stored as a map containing <key, value> pairs, and the *validateEvent()* can be used to check if a specific instance of an Event conforms to the definition in the Event Definition Database. There is no standard mechanism that we enforce on converting incoming notifications to Events within different Event Listeners. However, in case the incoming notifications have a fixed format/protocol, incoming notifications can be converted to Events by defining a rule set. For example, in the case of the GUI that is also developed in ONE, we can fix the mechanism for expressing notifications for the GUI, and can define rules, which are stored in the GUI Event Rules database to convert incoming notifications to Events.

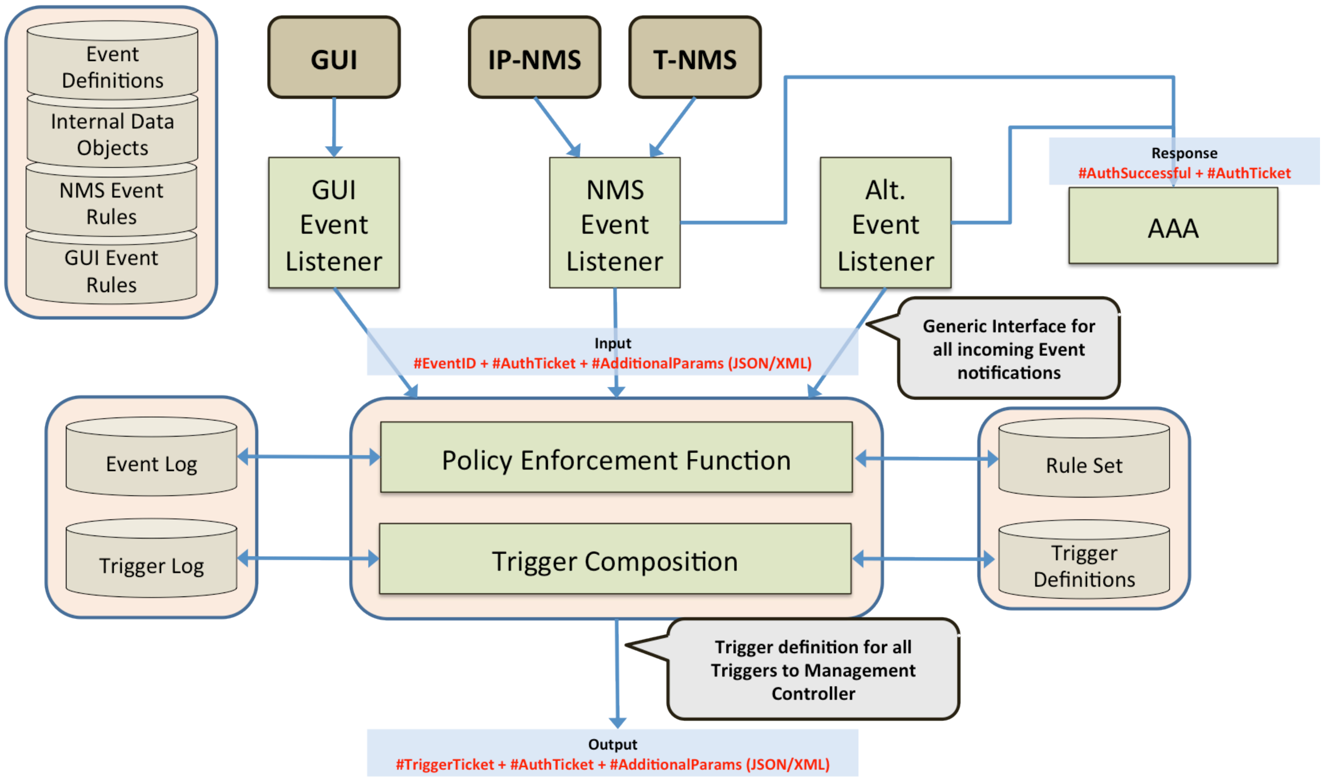

Event Definitions
Internal Data Objects
NMS Event Rules
GUI Event Rules
GUI
IP-NMS
T-NMS
GUI Event Listener
NMS Event Listener
Alt. Event Listener
Response
#AuthSuccessful + #AuthTicket
AAA
Generic Interface for all incoming Event notifications
Input
#EventID + #AuthTicket + #AdditionalParams (JSON/XML)
Event Log
Trigger Log
Policy Enforcement Function
Trigger Composition
Rule Set
Trigger Definitions
Trigger definition for all Triggers to Management Controller
Output
#TriggerTicket + #AuthTicket + #AdditionalParams (JSON/XML)

Figure 19. Trigger Module Components and their Interactions.

In our implementation, we developed two event listeners, both of which expose web service interfaces to receive events from the external systems. The GUI event listener interface (cf. Appendix B, Section 12.5.1) assumes that the authentication has already been performed by the GUI, and accepts the *EventID* and *AuthTicket* as fixed parameters while all other event related parameters are encoded as a JSON string. The events coming from the GUI can vary significantly, and therefore interface specification only accepts a string as input as new interfaces need not be implemented for every new event that is added to the ONE adapter. On the other hand, the NMS Event listener is designed mostly to respond to alarm or event traps coming from the network to the IP or the T-NMS, and the corresponding interface (cf. Appendix B, Section 12.5.2) includes parameters to categorize the types of alarms (*alarmClass*), the nature and severity of the alarm (*isServiceAffecting, alarmSeverity*) and the time while the alarm was raised in the network. Apart from the basic classification of the alarm itself, additional parameters such as topological information may be included within the event, and is included as additional parameters encoded as a JSON string within the interface.

```
Class Event {                              Class Trigger {
        //Variables                                //Variables
        String eventID;                            String triggerTicket;
        String authTicket;                         String authTicket;
        Map params;                                Map params;

        //Functions                                //Functions
        String getEventID();                       String getTriggerTicket();
        String getAuthTicket();                    String getAuthTicket();
        Map getParams();                           Map getParams();
        Boolean validateEvent();                   Boolean validateTrigger();
        String getParamsJSON();                    String getParamsJSON();
}                                          }

              (A)                                        (B)
```

Figure 20.Generic Event and Trigger Definitions in the ONE adapter.

After an event is generated in an Event Listener, it is converted into a standard format as shown in Figure 19, and is sent to the Policy Enforcement Function (PCF) using a standard interface for all event listeners (cf. Appendix B, Section 12.5.3). Note here that the incoming interface of the policy enforcement function only takes the *EventID* and the *AuthTicket* as fixed parameters and all additional parameters are encoded as a String using a flexible data object definition such as JSON [JSON01]. By passing additional parameters as a String, we ensure that the input interfaces for the Policy Enforcement Function need not be extended when new events are incorporated in the event listeners. This ensures that event definitions can be updated in runtime. The PCF is a stateless service that is invoked for an event arrival, which means that incoming event notifications are not queued for sequential processing. The choice of this ensures that the PCF can scale easily with increase in frequency of event arrivals at the PCF. However, this can pose a challenge in the way events are mapped to triggers. We describe these as two different scenarios.

Event Trigger Mapping

In case a trigger can only be generated by a specific (individual) event, when the PCF is invoked by an incoming event notification, the PCF does not require information about other events in the system. A classic example can be when events are only generated by a user GUI, where all operations (or workflows) in ONE can be triggered using a unique operation in the GUI. In this case, an incoming event is logged in the event log database, and then a rule set, stored in an external database, is applied on the incoming event to determine if the incoming notification should invoke an operation. If applied rules indicate that the incoming event notification should invoke a workflow, Trigger composition is initiated which is responsible for the composition of the trigger. The rules in the rule set also define how an event should be mapped to a trigger, and if additional static parameters should be inserted into the trigger.

The trigger definition within the Trigger module also uses a generic definition similar to an Event as shown in Figure 19(B). Trigger definitions are stored in the Trigger database, and are uniquely identified by a *triggerID*. Each workflow is uniquely associated to a trigger, so a generated trigger always invokes a specific workflow in the ONE adapter. In the definition of a trigger, we generate the *triggerTicket* using the *triggerID* and a unique counter, so that triggers for the same workflow at different times can be identified within ONE. The *triggerTicket* is used for monitoring the progress of operations initiated by a specific trigger instance across the ONE adapter. Similar to the Event structure, the Trigger also stores parameter in a Map structure, and uses the Trigger Definitions to validate triggers generated in the PCF. The validation provides an extra check to confirm if the rule set generates a valid trigger. After a trigger is generated, it is is logged in the Trigger Log database, and is then sent to the Management Controller. The interface specification is also flexible, as was seen in the PCF. This, along with the fact that the rule-set applied to the PCF is stored externally and can be modified in run time, ensures that new triggers can be dynamically introduced into the ONE adapter.

We now also briefly describe strategies in case **N:1 Event to Trigger** Mappings are used in the ONE adapter. This can be seen in case a series of automated events are required to initiate operation in the ONE adapter. A classic example can be in case of offloading: for example, consider a scenario where SNMP traps are issued when traffic on an IP link exceeds a specified threshold are used to initiate the offloading operation within ONE. It is possible that operators want to initiate offloading only when overloading is observed for a period of time, which can be represented as receiving multiple trap notifications in a specified time period, to ensure that a very short lived traffic spike does not initiate offloading. The same can be achieved using historical event information stored in the Event Log: In this case, on the arrival of an event at the PCF at time *t*, the PCF logs the event in the Event Log, and extracts all Events stored in the Event log with incoming times in the range of [*t*-Δt, *t*), and policy rules then take into consideration the historical event data as well as the incoming event to generate triggers. In the example of offloading, if an example of a policy was that at least 3 traps from the same link should have been received in the last Δt = 10 minutes, the PCF can use historical data 10 minutes previous to all event arrivals, and should be able to generate the trigger for offloading when the third trap arrives from the network.

Note that in this architecture, all checks are initiated by the arrival of an event, which is not sufficient for complex event correlations. A representative example is shown below: in a system with three events *a*, *b*, and *c,* we can have a scenario where trigger *x* should be generated if only *a* and *b* arrive within a specific time window Δt while trigger *y* should be generated when all of *a, b,* and *c* arrive in the specified time window. In case that event arrivals are in sequence – *a* followed by *b* followed by *c*, enforcement of accurate policies based on just historical data is not possible. A possible solution to the same could be to have an external timer, which generates a *fake event* that is scheduled for the future and which initiates policy enforcement only on historical data. So in the example, upon the arrival of *b*, a fake event would be generated in the future in order to wait a specified time to see if event *c* has arrived. When the trigger expires and the fake event is received by the PCF, the rule set would check if *c* also arrived: In case *c* arrived, the PCF on arrival of *c* would have generated the trigger *y*, and the PCF would do nothing. However, in case *c* did not arrive, the PCF could then accurately generate the trigger *x*.
It should be noted however that we believe that such event notifications would incorporate significant complexity in defining the rule set for the PCF validating the same. We therefore, in our

design, will propose to go for simple policies for trigger generation and incorporate the complexity for choosing the correct action within the workflow definitions. We also note that automated event notifications from network devices (for example SNMP traps) often need to be correlated along with topology information in order to provide context: for example, in case of a fiber cut, multiple SNMP traps will be generated for different lightpaths that traverse the fiber, and in order to identify the correct failure, root cause analysis along with topology information is required. As existing NMSs already possess these features, in our architecture, the IP and the T-NMS also have an Event Generation Module, which analyses SNMP traps/notifications coming from the network, and if required, generates an Event Notification which is sent to the Trigger Module. Also note that the IP and T-NMSs can make use of the Ontology Mapper to translate notifications specific to different device vendors into a standard vendor-independent event notification within the ONE adapter.

Trigger Module Administration

The administration of the trigger module consists primarily of Event, Trigger and Policy administration functionalities.

In Section 12.5.4.1 the interface specification of the event administration is shown. It is possible to retrieve all Event or specific Events, or add and delete an event. For every operation, an AuthTicket is necessary to ensure that the operation is executed only by an authorized user. The interfaces for administration of triggers offer similar interfaces (cf. Section 12.5.4.2). The event and trigger specifications are used by the trigger modules to validate incoming events and generated triggers before they are sent to the management controller. As specified before, the PCF uses a policy to map events to triggers, and this policy can be administered by the policy administration interfaces (cf. Section 12.5.4.3). The policy is assumed to be expressed as a string, and can be retrieved or updated using these functions.

5.2 IP NMS-Control Module

The IP-NMS Control Module is a required auxiliary module of the ONE adapter. This module’s main role is to act as interface between the IP network and the ONE adapter where two scenarios are possible:

1. The first scenario corresponds to the case when there is an external IP Network Management System already present in the operator’s network, which is capable of automatically performing the configuration of NEs at the IP layer. In this case, the IP NMS control module will have to cooperate with the ontology mapper to handle in the correct language the communication with each IP-NMS available in the IP network.

2. The second scenario corresponds to the case when there is either no external IP-NMS or the ones that might be available to mediate between the ONE adapter and the IP nodes do not provide support to automatically configure the NEs at the IP layer. This case also needs

cooperation with the ontology mapper to adapt the commands and communication depending on the vendor and software version of each IP router.

The interfaces shown in Figure 21 represent the two previously mentioned use cases. Inside the IP-NMS module, two sub-modules can be seen, each responsible for the processing of the incoming information, sharing a common access to Config and Status of the service components started by the parser and pre-processor.

Both blocks have access to the web-services to communicate with the corresponding external actor (IP NMS web service and NETCONF, SNMP or CLI in the case that no IP-NMS is available).

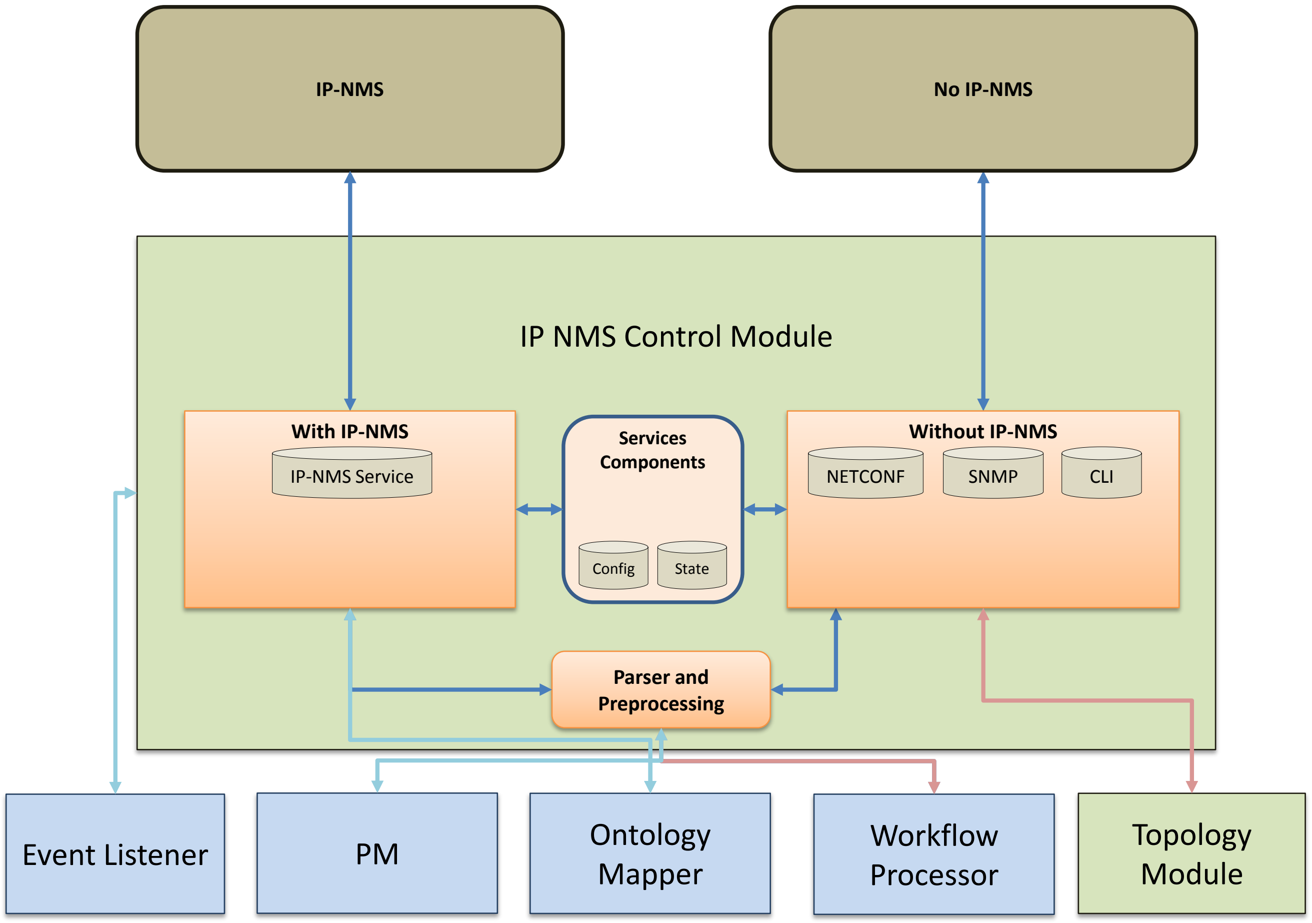


Figure 21. Detail view of IP-NMS Interfaces.

The behaviour and algorithms will depend on the specific destination of the action requested to the IP-NMS control module.

The internal event listener is (front end of the parser and pre-processer) responsible for listening for incoming events generated by internal modules, such as the Workflow Processor or the Topology Module. The parser is then responsible for analysing the incoming event, in order to determine through a simple pre-processing stage if the current input is a configuration or a state retrieval request.

Once the decision is taken, the Parser and Pre-processer adapt the request depending on if there is an IP-NMS or not involved in the solution of the action.

5.2.1 Functionality in presence of an external IP-NMS

The IP-NMS control module (in presence of an external IP-NMS, i.e. Magalia) will make use of the specific WS adapted to the corresponding NMS. In case of more than one IP-NMSs there will be multiple WS for each of them.

Figure 22 depicts the workflow for GET/SET operations in the context of an external IP-NMS.

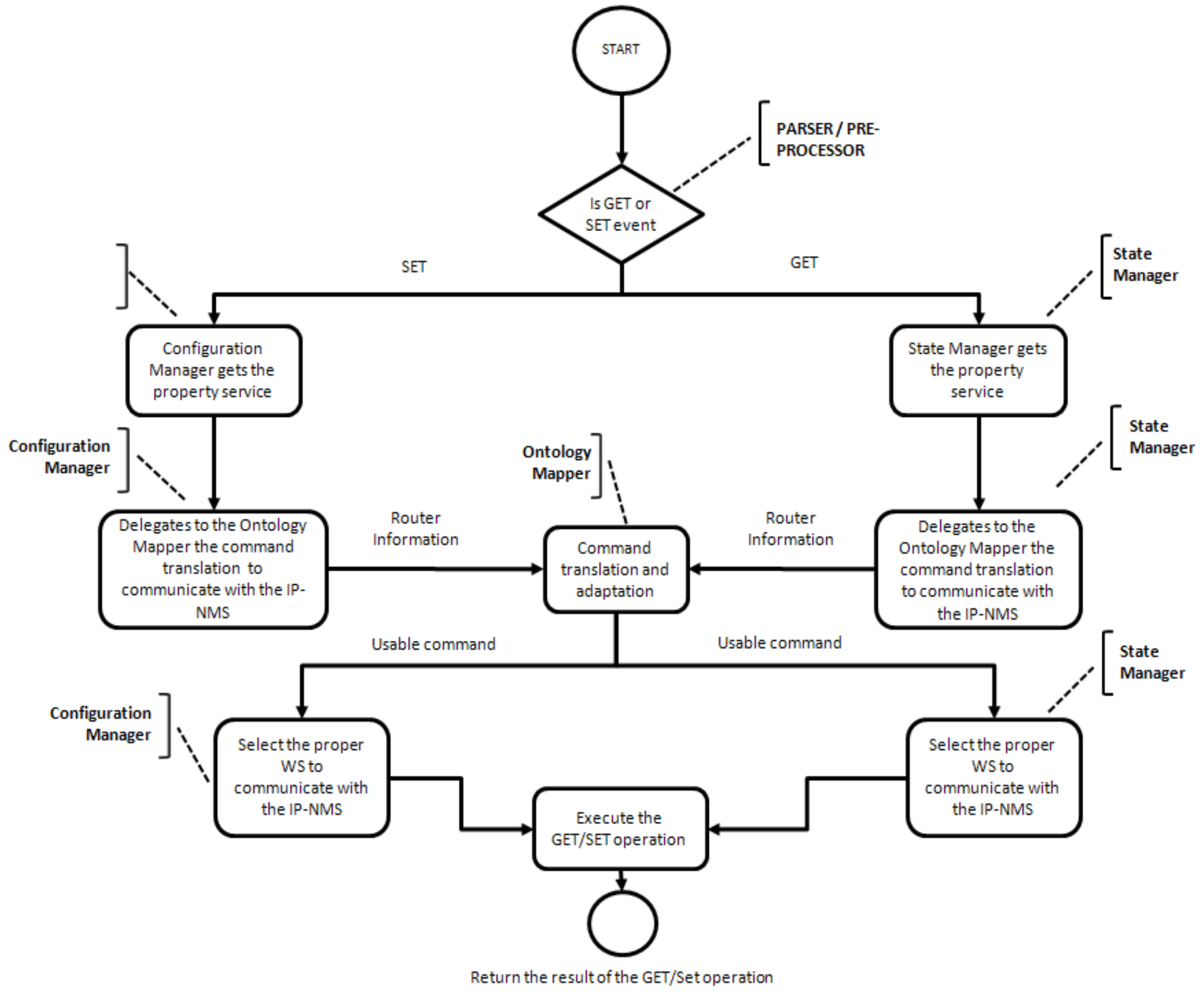

START
PARSER / PRE-PROCESSOR
Is GET or SET event
SET
GET
State Manager
Configuration Manager gets the property service
State Manager gets the property service
Configuration Manager
Ontology Mapper
State Manager
Delegates to the Ontology Mapper the command translation to communicate with the IP-NMS
Router Information
Command translation and adaptation
Router Information
Delegates to the Ontology Mapper the command translation to communicate with the IP-NMS
Usable command
Usable command
State Manager
Configuration Manager
Select the proper WS to communicate with the IP-NMS
Execute the GET/SET operation
Select the proper WS to communicate with the IP-NMS
Return the result of the GET/Set operation

Figure 22. Workflow for GET/SET operations in the context of an external IP-NMS.

The pseudo-code for the functionalities of the IP NMS in presence of an external IP-NMS is the following:

```
Input:    (Action to perform: (Get, Set), Details of the action (Set interface IP, Node IP, Get Router Info), Affected Entity
(Router X))
Output:  (Result of the Action, Details of the result)

If (Action == Get):
          To state manager (Details of the action, Affected Entity)
Else:
          To configuration manager (Details of the action, Affected Entity)

State manager
          If (Check action feasibility (consistence between request and knowledge) == true)

     Commands = RequestOntologyMapperCommands (Details of the Action, entity information)
                    WS = GetWSExtIPNMS (AffectedEntity)
                    Action Result = WS (Commands)
                    Return Action Result
  Else:
     Return Action Failure

Configuration manager
          If (Check action feasibility (consistence between request and knowledge) == true)

     Commands = RequestOntologyMapperCommands (Details of the Action, entity information)
                    WS = GetWSExtIPNMS (AffectedEntity)
                    Action Result = WS (Commands)
                    If (Action Result == Success)
                              Refresh Configuration Action Database
                    Return Action Result
  Else:
     Return Action Failure
```

5.2.2 Functionality without an External IP-NMS

The IP NMS control module needs to interact with the Ontology Mapper that is translating the commands used by the external IP-NMS.

Interactions to external actors are solicited actions, which mainly include:

1. Getting information from external IP Network Elements requesting information through Network Management Station (IP-NMS) handling management information for all the devices under its domain.
2. Setting IP Network Elements configurations through an external IP-NMS.

The IP-NMS control module, in case there are no external IP-NMS, will make use of the standard communication protocols NETCONF or SNMP for GET and SET operations. It also adds the possibility of using CLI in case the IP equipment has no support of specific actions through standard protocols.

Figure 23 depicts the workflow for GET/SET operations in case there is no IP-NMS. As it can be seen, there is a small difference between both IP-NMS and no IP-NMS behaviours because the translation part relies on the Ontology Mapper. The no IP-NMS actions will be performed by the available protocols (SNMP, CLI and NETCONF).

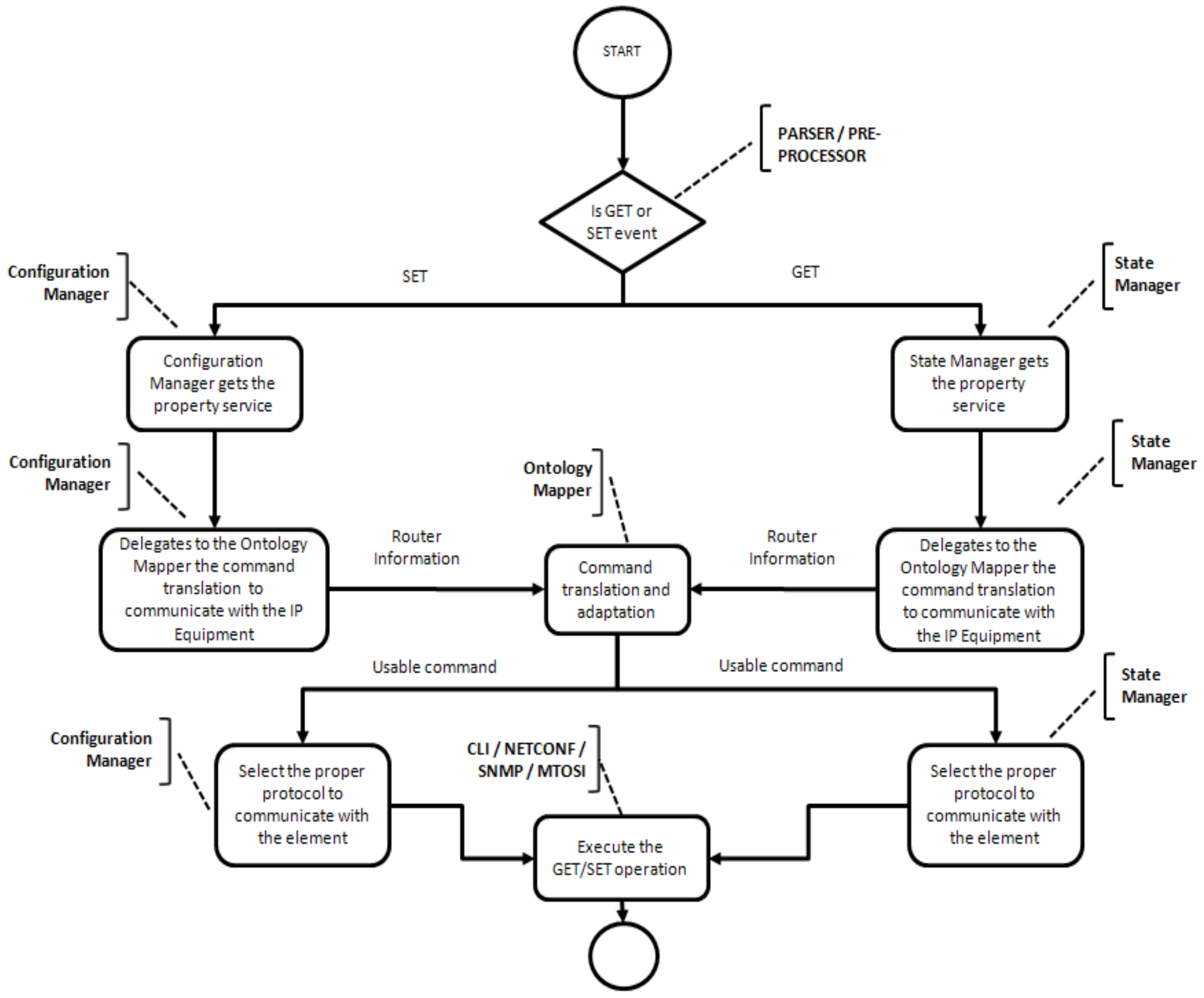

START
PARSER / PRE-PROCESSOR
Is GET or SET event
SET
GET
Configuration Manager
State Manager
Configuration Manager gets the property service
State Manager gets the property service
Configuration Manager
Ontology Mapper
State Manager
Delegates to the Ontology Mapper the command translation to communicate with the IP Equipment
Router Information
Command translation and adaptation
Router Information
Delegates to the Ontology Mapper the command translation to communicate with the IP Equipment
Usable command
Usable command
State Manager
Configuration Manager
CLI / NETCONF / SNMP / MTOSI
Select the proper protocol to communicate with the element
Execute the GET/SET operation
Select the proper protocol to communicate with the element

Figure 23. Workflow for GET/SET operations in case there is no IP-NMS.

In summary, the differences between the instances are the following:

1. The CLI inclusion in case of not having an IP NMS can be used to communicate with the IP Nodes as a way of configuration and consulting information.
2. SNMP will only be available to get information from the IP Nodes in case of no IP NMS present.
3. As in the case of CLI for the IP Nodes direct configuration, the MTOSI web services are available in case of communicating with an IP NMS.

The rest of the functional blocks are common to both instances:

1. Split the request into GET & SET operations (Parser/Pre-Processing Block)
2. Select the adequate Web Service to perform the requested operation (Configuration Manager/ State Manager to Web Services Block).
3. Call the Ontology Mapper Module.
4. Communicate with the corresponding protocol to the external actor (NETCONF / SNMP / CLI / MTOSI).
5. Return the answer of the process to the Workflow Processor / Topology Module.

5.2.3 Interface with PM

The interface with the Performance Monitoring module is defined to allow the PM request directly from the IP layer bandwidth information. A simple query from the PM for bandwidth from an IP interface will be returned with the bps information read from SNMP or the monitoring tool the IPNMS uses in each case.

5.2.4 Administration

The IP-NMS control module needs information to be defined in order to work properly. Since some monitoring operations are done by the IP-NMS Control Module, the traffic threshold has to be defined in the working links to be capable of triggering offloading events.

The logging and configuration protocol information when connecting nodes is also suitable to be changed. As a consequence, this is reflected also in the administration interface (Section 12.6).

5.2.5 Interface with NMS Event Listener

The IP NMS control module can listen to incoming SNMP events/traps from the IP network, and some of these events can be configured to initiate operations in the ONE adapter. Here, the IP NMS can be configured so as to choose which events in the network should initiate an operation in the ONE adapter, and upon each such event, the IP NMS Control module initiates operation by invoking the NMSEventListener in the Trigger Module (cf 12.5.2) which then follows the same execution flow as other (operator generated) events in the ONE adapter.

5.3 Transport NMS Control Module

The Transport NMS control module is responsible for providing interconnectivity between the ONE adapter’s modules and external Transport NMS (T-NMS) system. Together with the Ontology Mapper Module it is supposed to create abstraction level for the external T-NMS system present in the operator’s environment, and give access to required operations to the adapter’s internal modules.

Several functions have been recognized as required by ONE’s internal modules:

- Topology Retrieval

  – provides internally requested topology information from external T-NMS.

- Service Configuration

  – commands external T-NMS to create a service between the endpoints. If multiple domains governed by separate T-NMS systems are engaged in the service, then each of them should get a set of path fragments creating all together the requested service.

- Get PM data

  – collects performance monitoring records requested by measurement module

All the workflows involving communications between the Transport NMS Control Module and external T-NMS systems have to pass previously through Ontology Mapper Module, in order to be adapted to the T-NMS specific format.

T-NMS Control Module is responsible for proper treating of the notifications coming from T—NMS. It should involve recognition of notifications, update of internal ONE Adapter topology data, and forwarding of relevant notifications to the Trigger Module. Figure 24 shows the main interactions of the NMS Control Module.

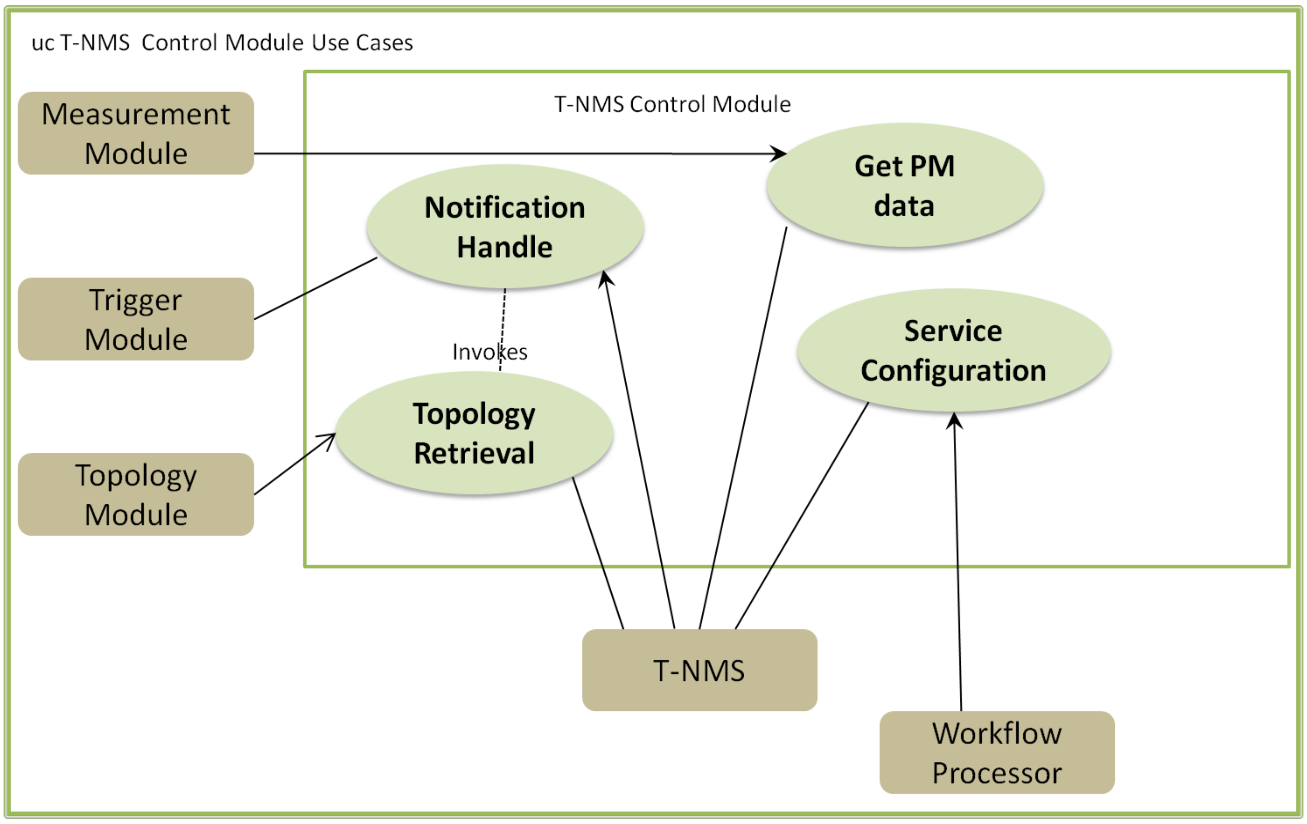


Figure 24. The T-NMS Control Module and its main interactions with other modules and external systems.

5.3.1 Notification handling

NMS Control Module has to register itself as a recipient of notifications from the managed NMS. T-NMS Control Module needs to recognize the type and impact of each notification received. In general two types of notifications should be handled. The ones that can be mapped into the ONE's Adapter internal notification type, they have to be enriched with location reference, and forwarded to the Trigger Module over the NMSEventListener interface (cf 12.5.2). All the rest of notifications have to be recognized for the possible alteration of topology. If the notification informs on topology changes, the Topology Module needs to be notified, and updated with the changes. If notification does not concern topology, it shall be only logged, and discarded. The interactions described in the previous paragraphs are shown in Figure 25.

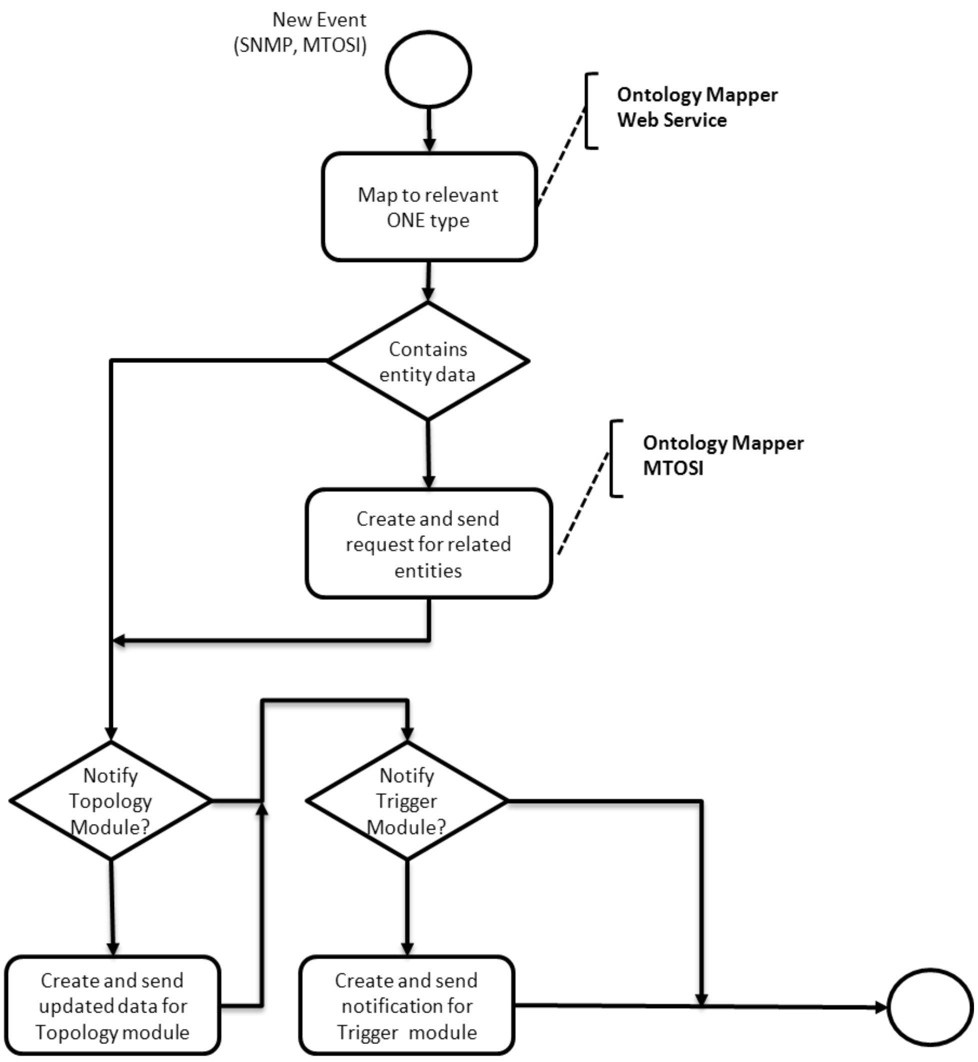


Figure 25. T-NMS Control Module notifications handling.

5.3.1.1 SNMP Binding Connector

Binding Connector uses Simple Network Management Protocol for getting alarms from a node. Firstly request for current conditions is performed and retrieval of related entities and then connector starts listening for incoming traps. Whole process was illustrated on Figure 26. Traps must be mapped and correlated with ONE Topology and then sent to Trigger Module.

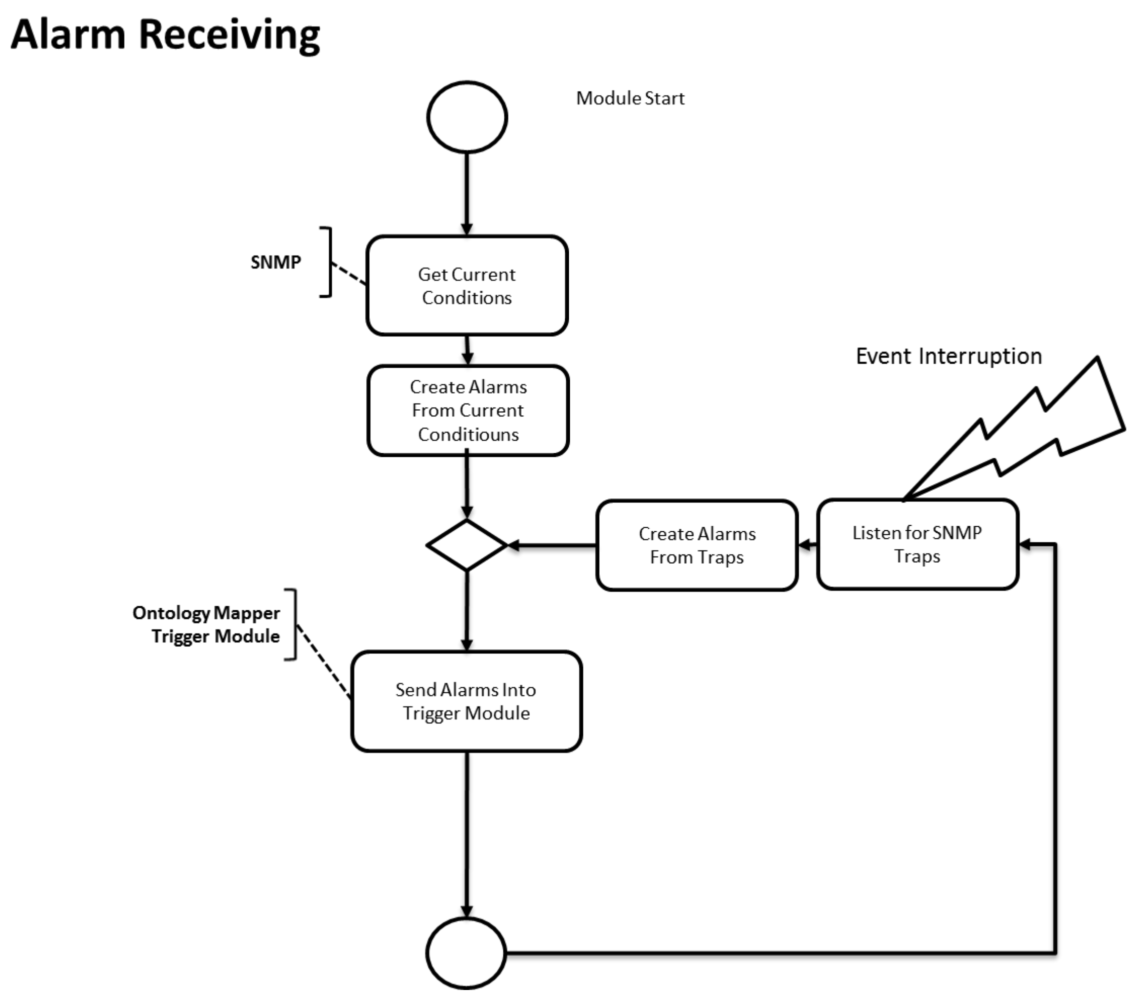


Figure 26 Flow Chart illustration of alarm receiving process.

5.3.2 Service Configuration Interface

Service or path related operations that are relevant to the ONE adapter are:

- Service creation.
- Service deletion.
- Verification of possibility of service creation.

All are presented at the flowcharts below, showing utilization of MTOSI 2.x interface to fulfil those operations. The ONE Adapter architecture doesn't however limit T-NMS Control Module to usage of MTOSI interface as a medium for communication with external T-NMS system. Any interface can be utilized; however logic that would need to be implemented as handling of the required operations would be more extensive.

Service creation (Figure 27) is made of two steps. Firstly requester checks whether path is correct and routable and then does creating actual service. This piece of software has to use MTOSI 2.x interface because MTOSI 1.x does not contain service configuration methods.

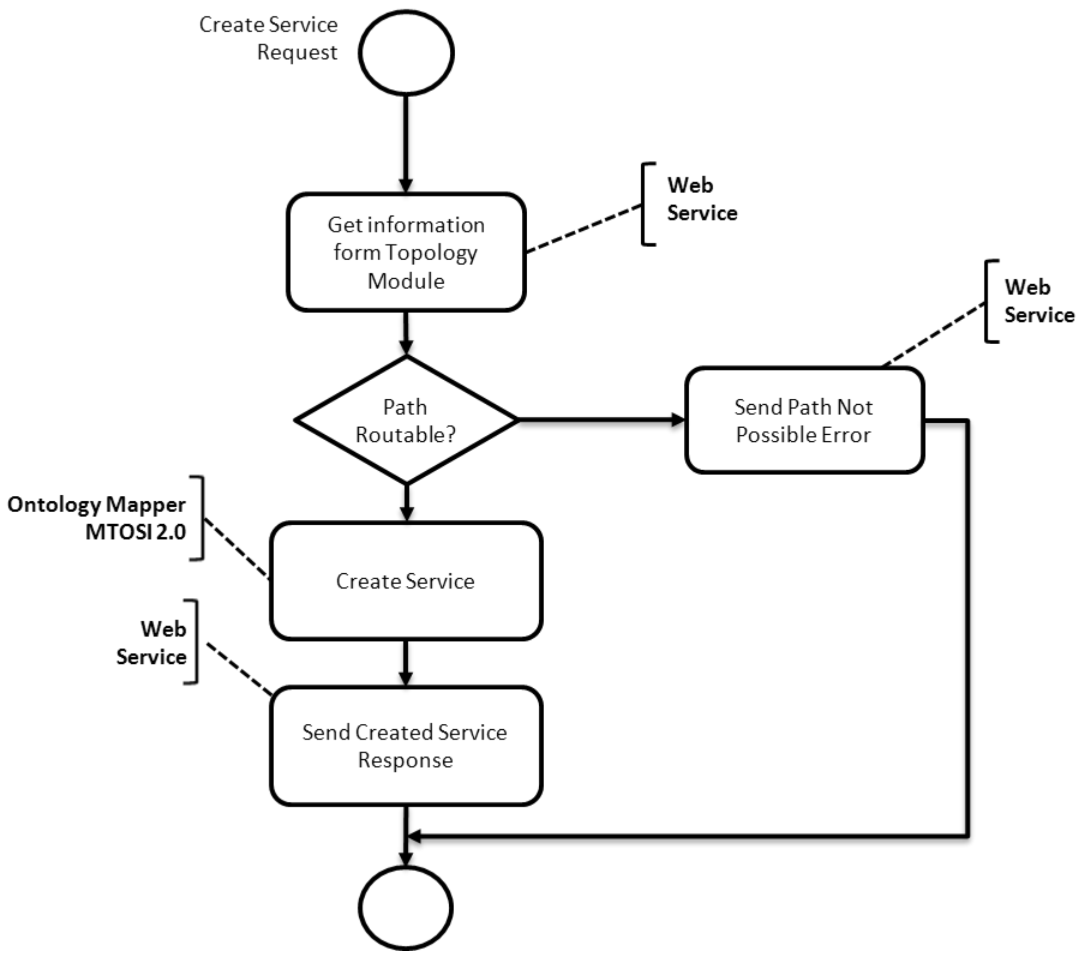

Create Service Request
Get information form Topology Module
Web Service
Path Routable?
Send Path Not Possible Error
Web Service
Ontology Mapper MTOSI 2.0
Create Service
Web Service
Send Created Service Response

Figure 27. T-NMS Control Module service creation.

Part of this operation flow is containing the second required operation: verification of possibility of service creation (Figure 28). Request with the endpoints and list of restrictions is being processed, and passed to external T-NMS. As a result the answer if path routing is possible is given, plus the list of paths of this route.

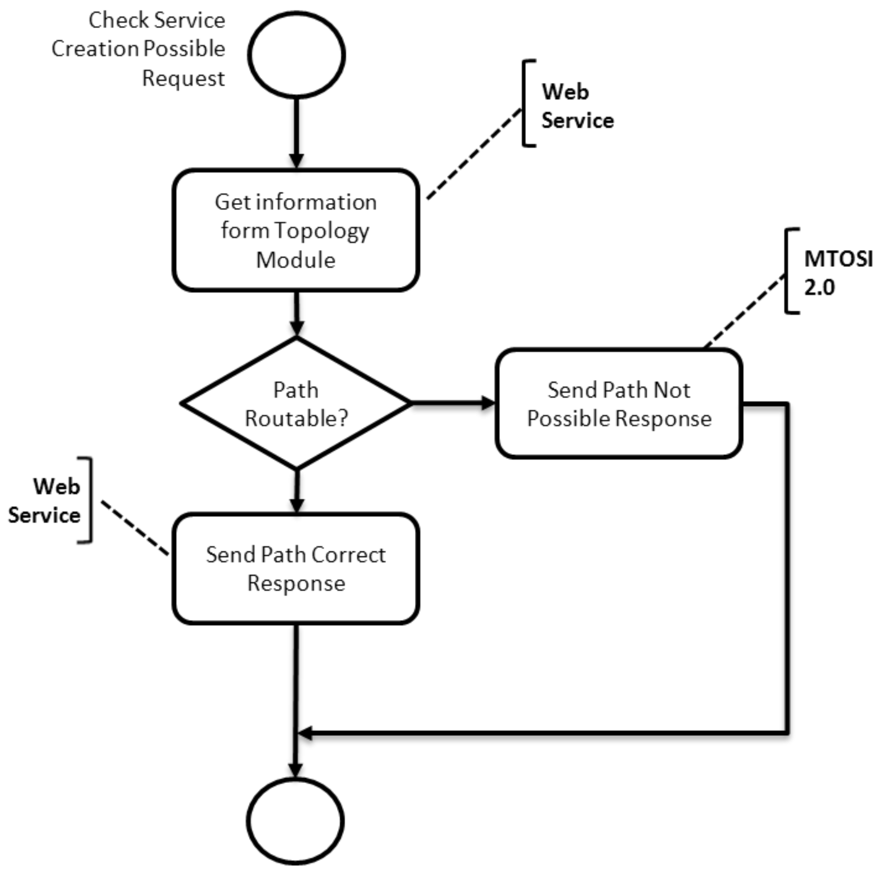

Check Service Creation Possible Request
Web Service
Get information form Topology Module
MTOSI 2.0
Path Routable?
Send Path Not Possible Response
Web Service
Send Path Correct Response

Figure 28. T-NMS Control Module service creation possibility verification.

Service removal (Figure 29) takes care of deleting the route within the transport network. As every action around paths, it should make sure to keep Topology Module updated on topology of the system.

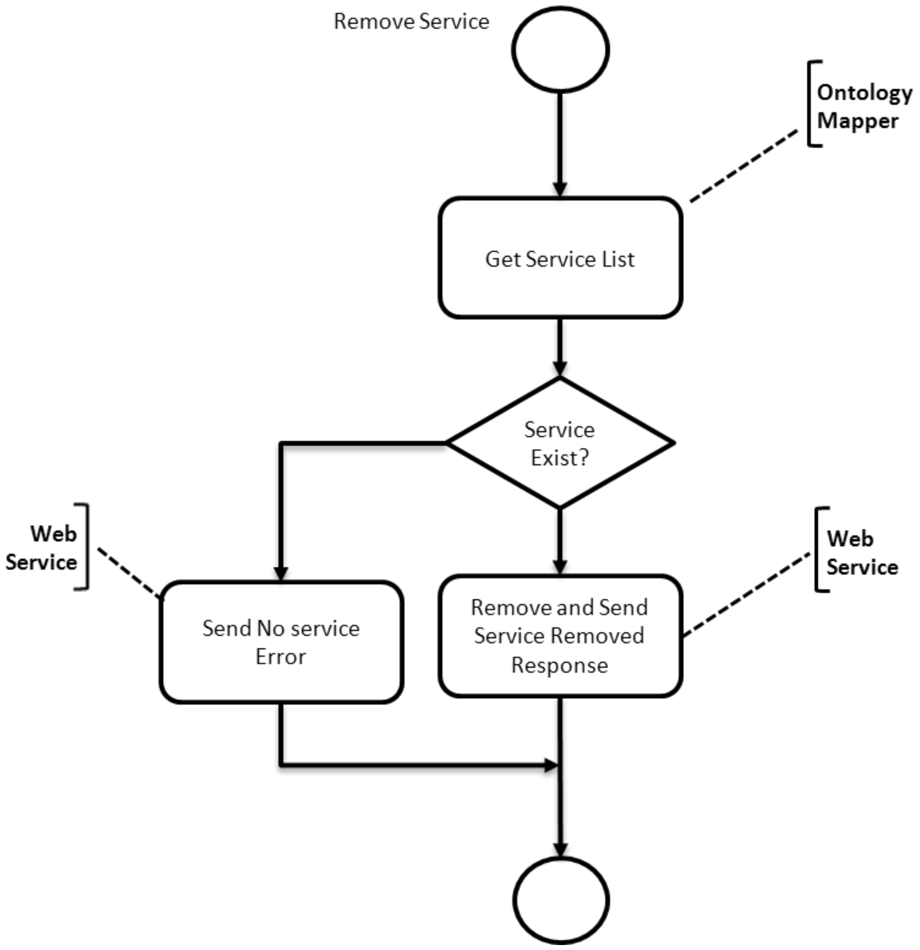
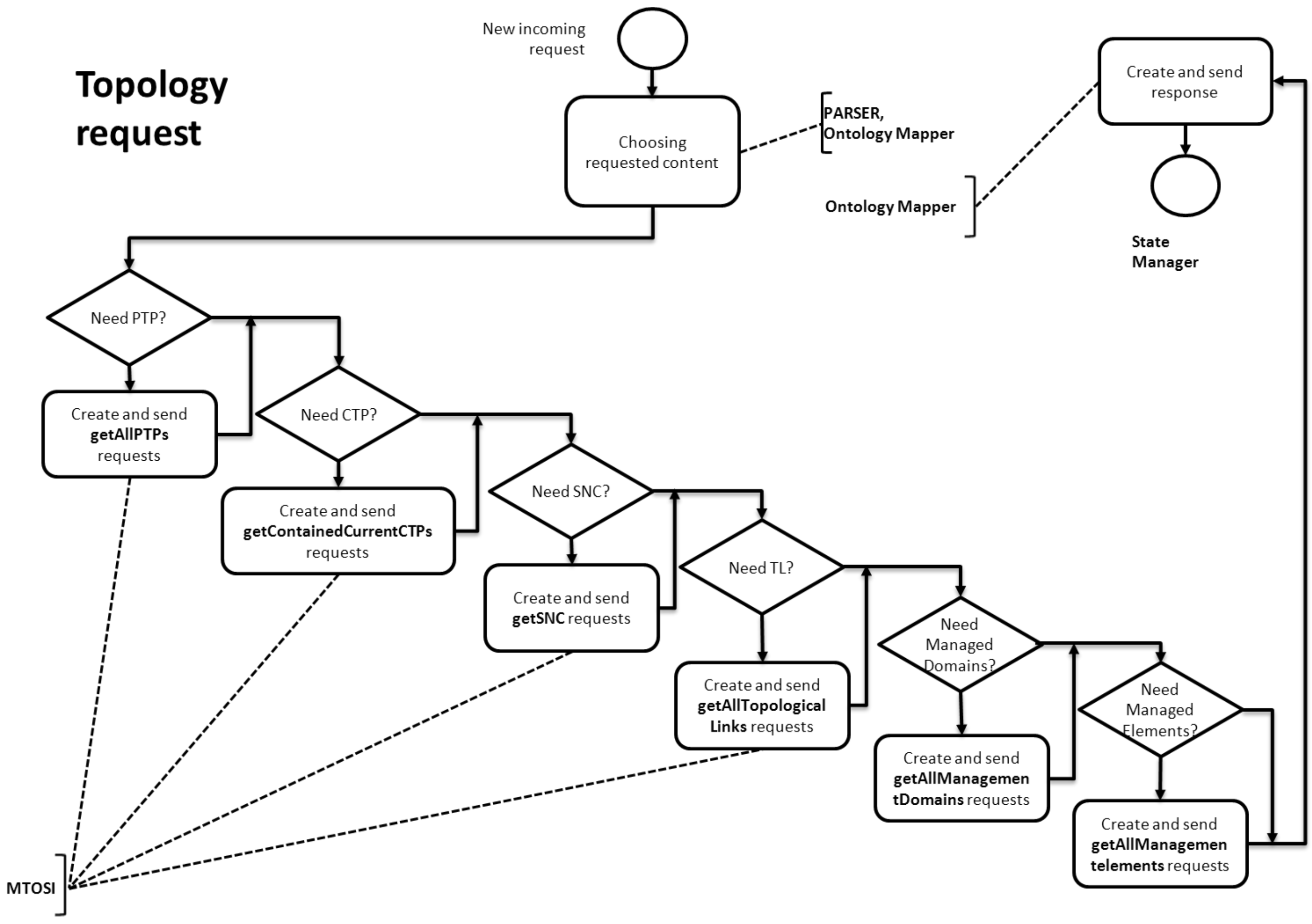


Figure 29. T-NMS Control Module service removal.

5.3.3 Topology Retrieval Interface

Topology retrieval is made of several smaller requests. All data have to be mapped into right objects and sent into Topology Module. This module can work with any version of MTOSI interface utilizing Ontology Mapper. This interaction is shown in Figure 30.



Topology request
New incoming request
Choosing requested content
PARSER, Ontology Mapper
Ontology Mapper
Create and send response
State Manager
Need PTP?
Create and send getAllPTPs requests
Need CTP?
Create and send getContainedCurrentCTPs requests
Need SNC?
Create and send getSNC requests
Need TL?
Create and send getAllTopological Links requests
Need Managed Domains?
Create and send getAllManagementDomains requests
Need Managed Elements?
Create and send getAllManagementelements requests
MTOSI

Figure 30. T-NMS Control Module topology retrieval.

5.3.4 Get PM Data Interface

Getting PM Data tends to be called only from Measurement Module. Requests are to be forwarded to proper network elements and after receiving performance monitoring data packed into response structure. Due to MTOSI concept all calls have to concern low level entities, and as such need translation from internal ONE adapter topology that is referenced in the request into the MTOSI specific entity. This interaction is shown in Figure 31.

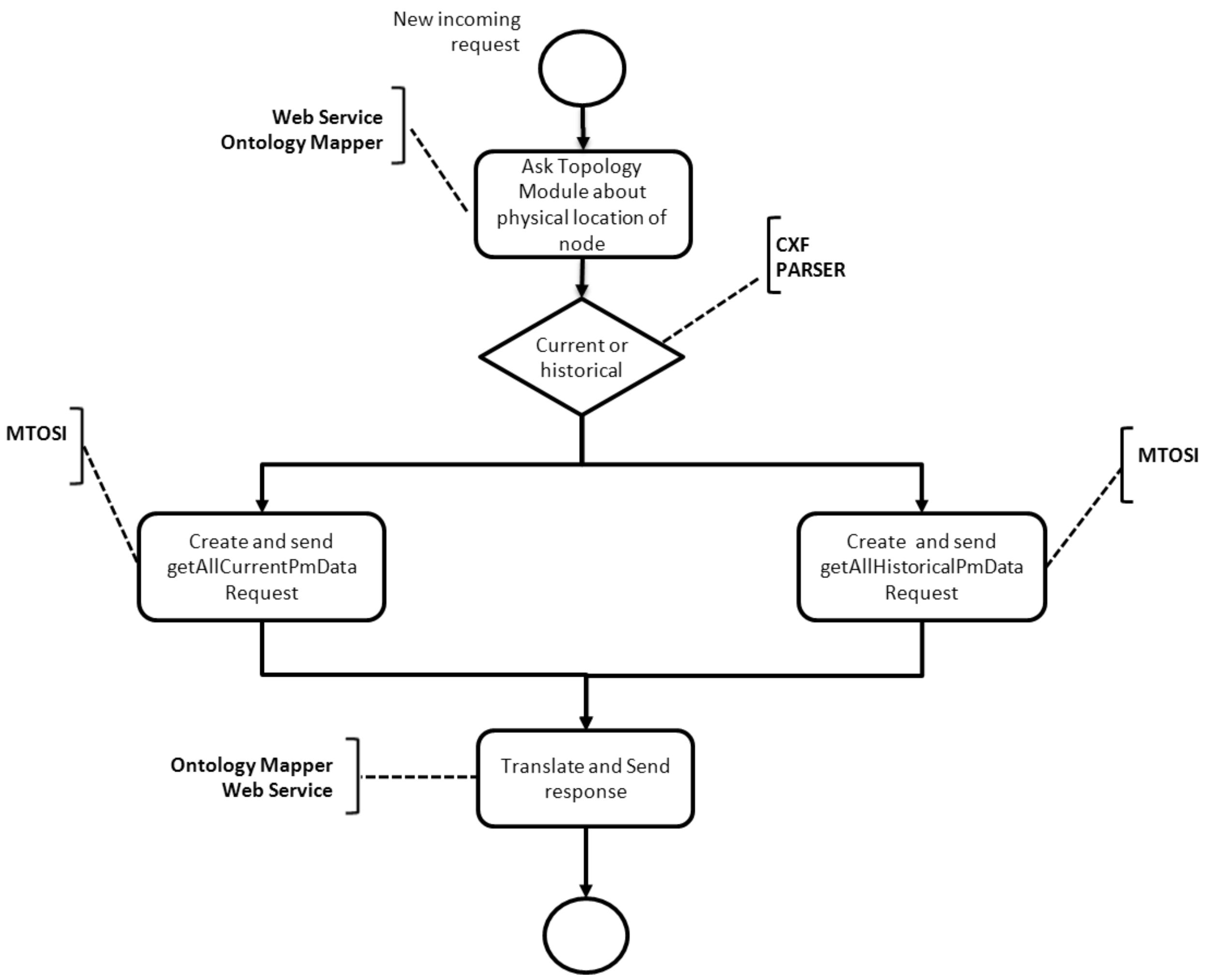

New incoming request
Web Service Ontology Mapper
Ask Topology Module about physical location of node
CXF PARSER
Current or historical
MTOSI
MTOSI
Create and send getAllCurrentPmData Request
Create and send getAllHistoricalPmData Request
Ontology Mapper Web Service
Translate and Send response

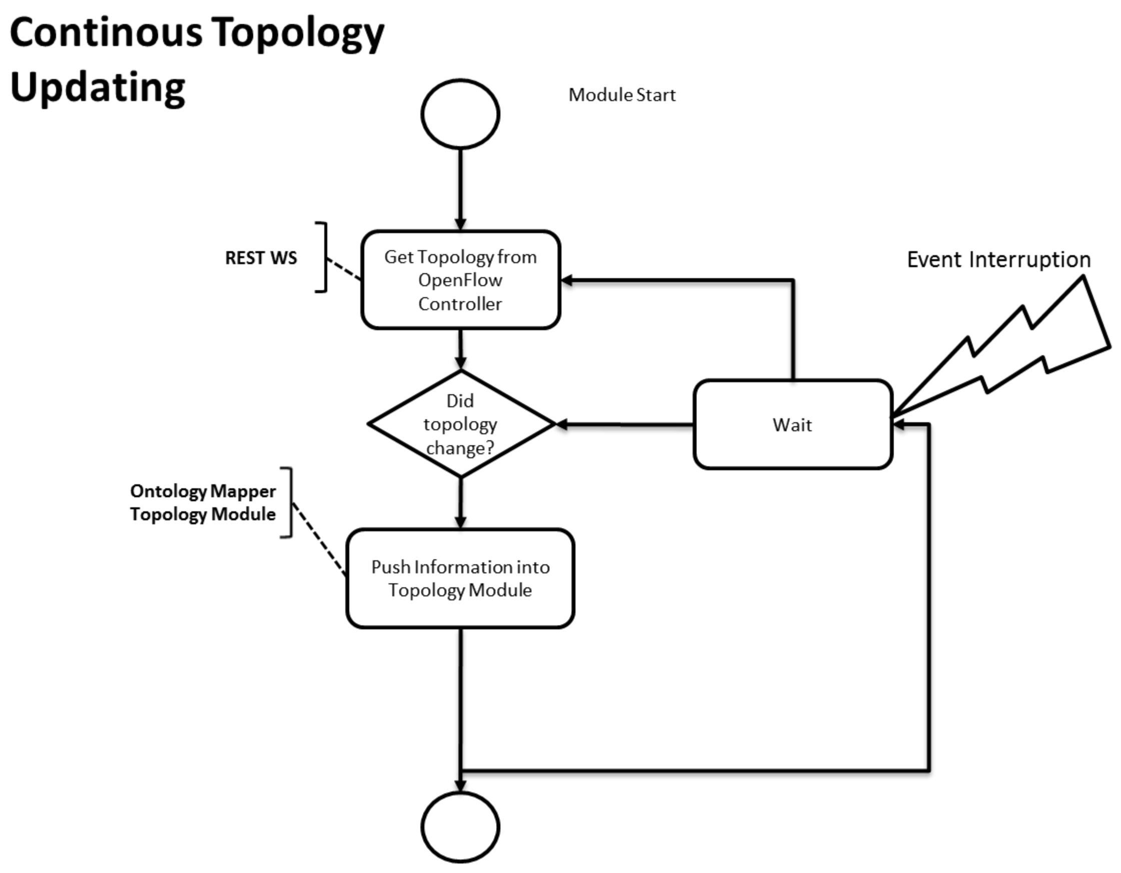
Figure 31. T-NMS Control Module performance data request.

5.4 OpenFlow Control Module

OpenFlow Control Module is a bridge between the ONE Adapter and the OpenFlow controller. It communicates through SOAP Web Service with other ONE Adapter's modules.

5.4.1 OpenFlow Topology Updating

OpenFlow does not have by default an asynchronous publish/subscribe service. The ONE Adapter requires up to date topology information that's why OpenFlow Control Module must periodically check for changes inside network topology. Also if certain conditions have occurred, additional topology retrieval should be performed. Process of gathering topology information is described on Figure 32.



Continous Topology Updating
Module Start
REST WS
Get Topology from OpenFlow Controller
Event Interruption
Did topology change?
Wait
Ontology Mapper Topology Module
Push Information into Topology Module

Figure 32 Flow diagram for Continuous Topology Updating.

5.4.2 Flow creation and removal

After mapping interfaces into correct switches and ports OpenFlow Control Module can perform operation through static flow pusher API. Whole process is shown on Figure 33 and Figure 34.

Due to lack of asynchronous service creation interface OpenFlow CM must periodically poll for the new flow information in the flow table, and if connection will not appear within specific timeout, module will assume service creation as not successful.

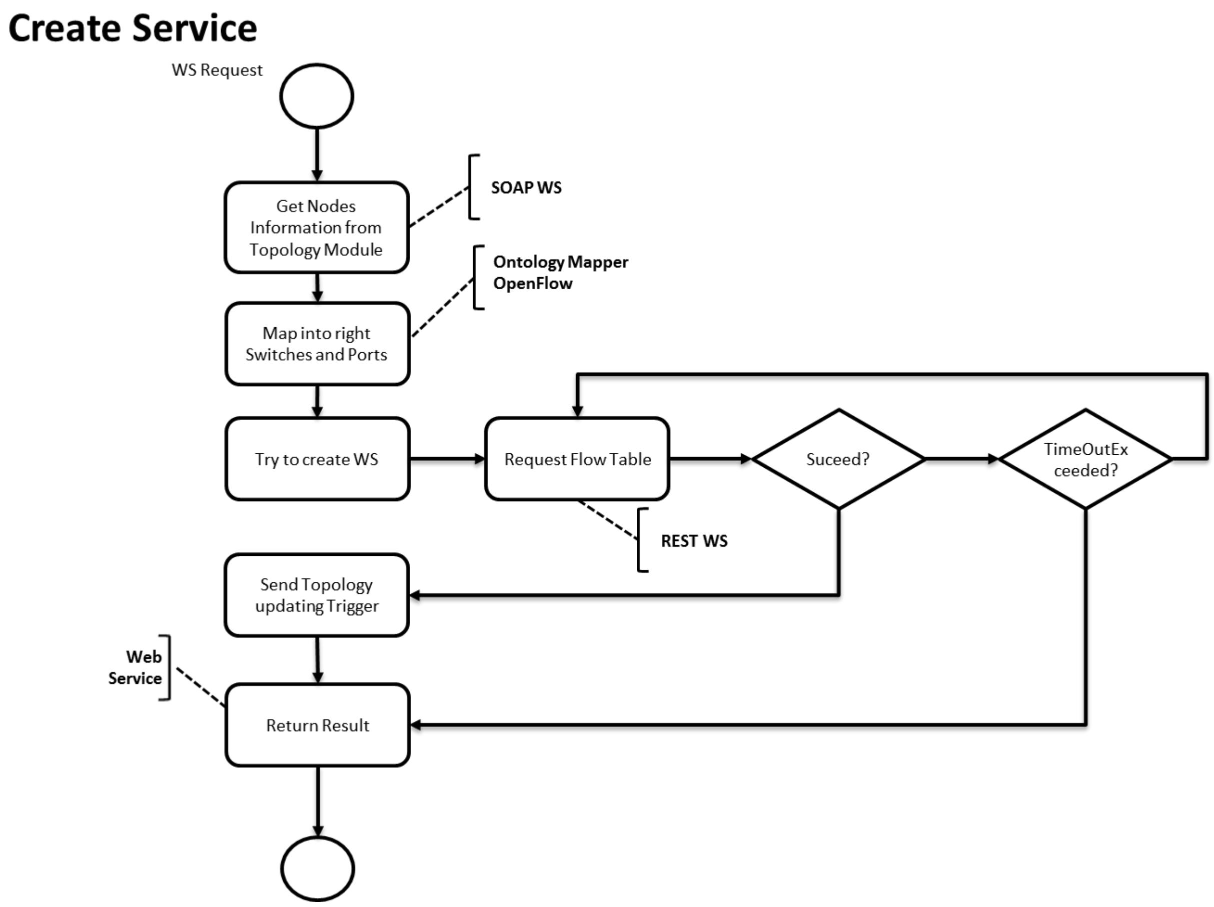


Figure 33 Create Service process ilustrating flow chart.

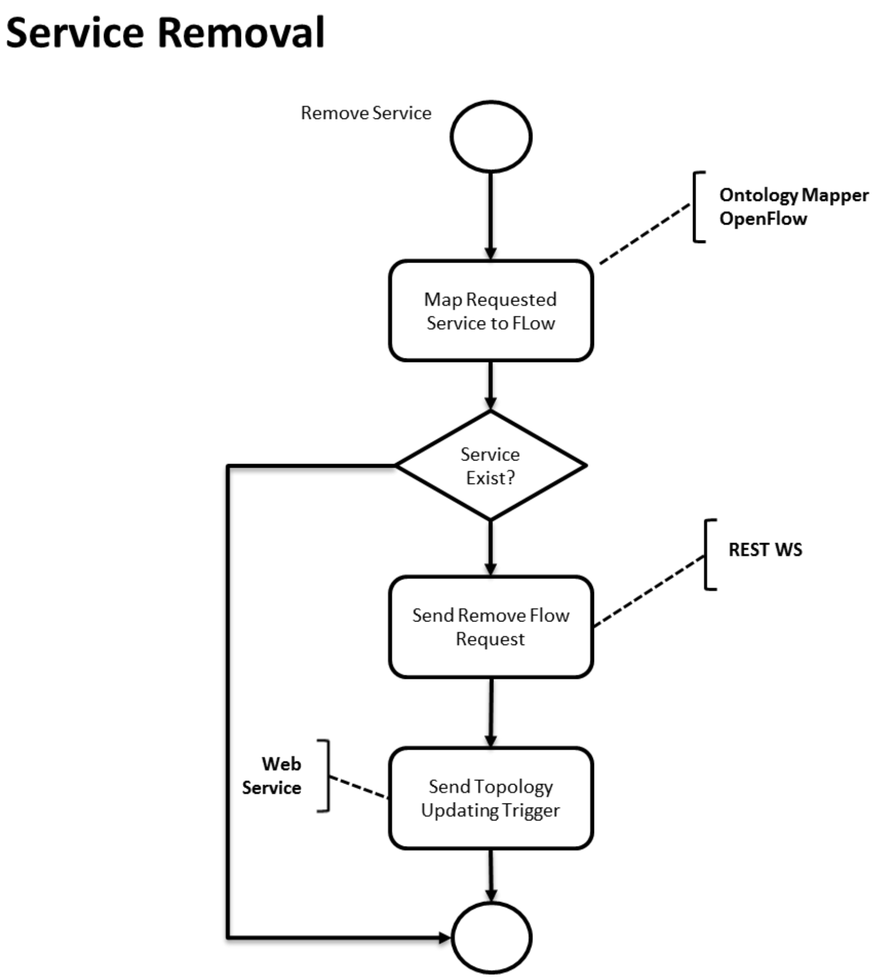


Figure 34 Remove Service process illustrating flow chart

5.4.3 Performance Monitoring Data Retrieval

Counters provide functionality of performance monitoring. Counters are maintained for each row table, flow entry, port, etc. Concept of this process is illustrated on Figure 35

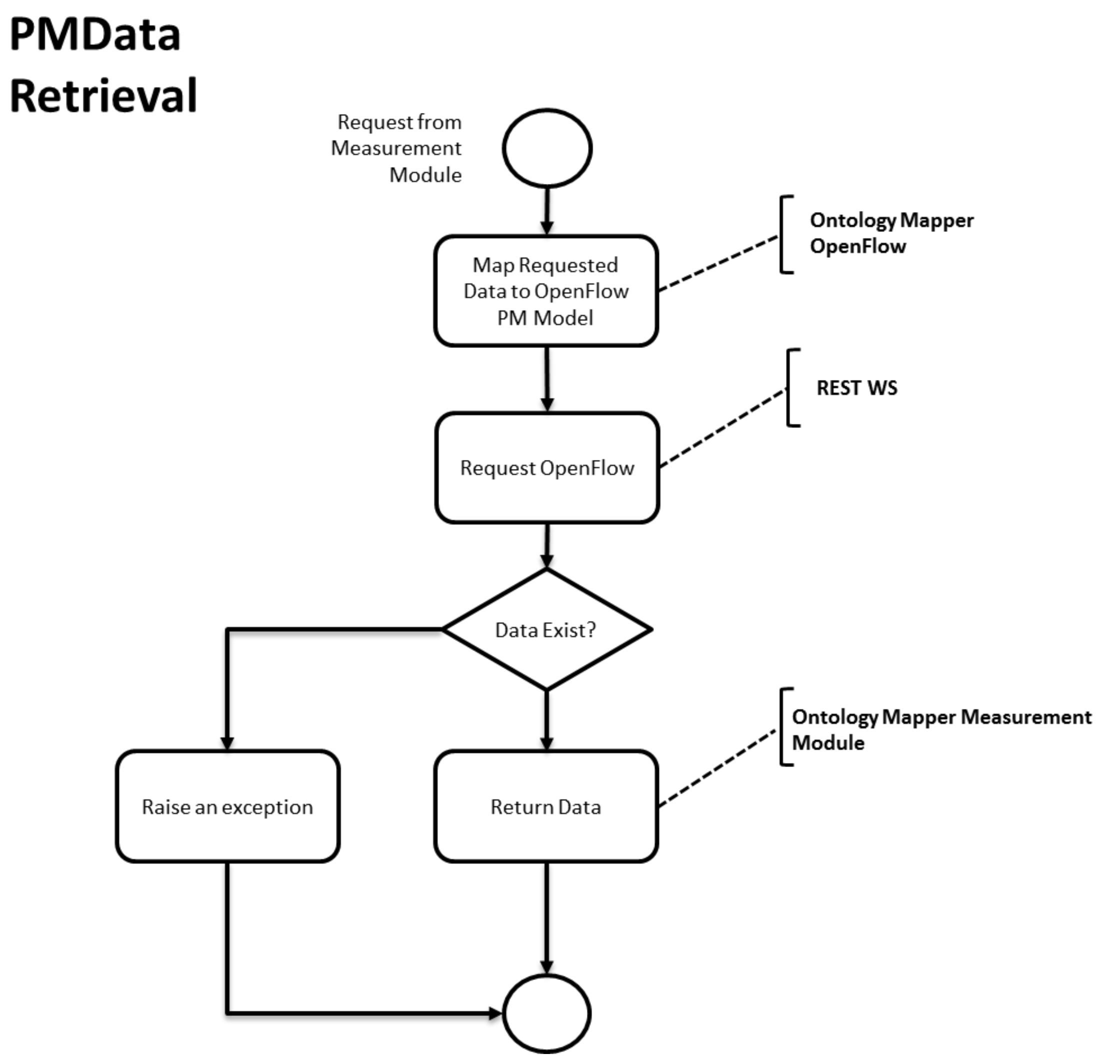

PMData Retrieval
Request from Measurement Module
Map Requested Data to OpenFlow PM Model
Ontology Mapper OpenFlow
Request OpenFlow
REST WS
Data Exist?
Raise an exception
Return Data
Ontology Mapper Measurement Module

Figure 35 Get PMData Service process illustrating flow chart.

5.5 Topology Module

The Topology Module is responsible for storing and providing network topology information, both per-layer topologies as well as inter-layer topology, and providing such topological information to the ONE adapter's internal modules. The internal modules can be seen in Figure 36.

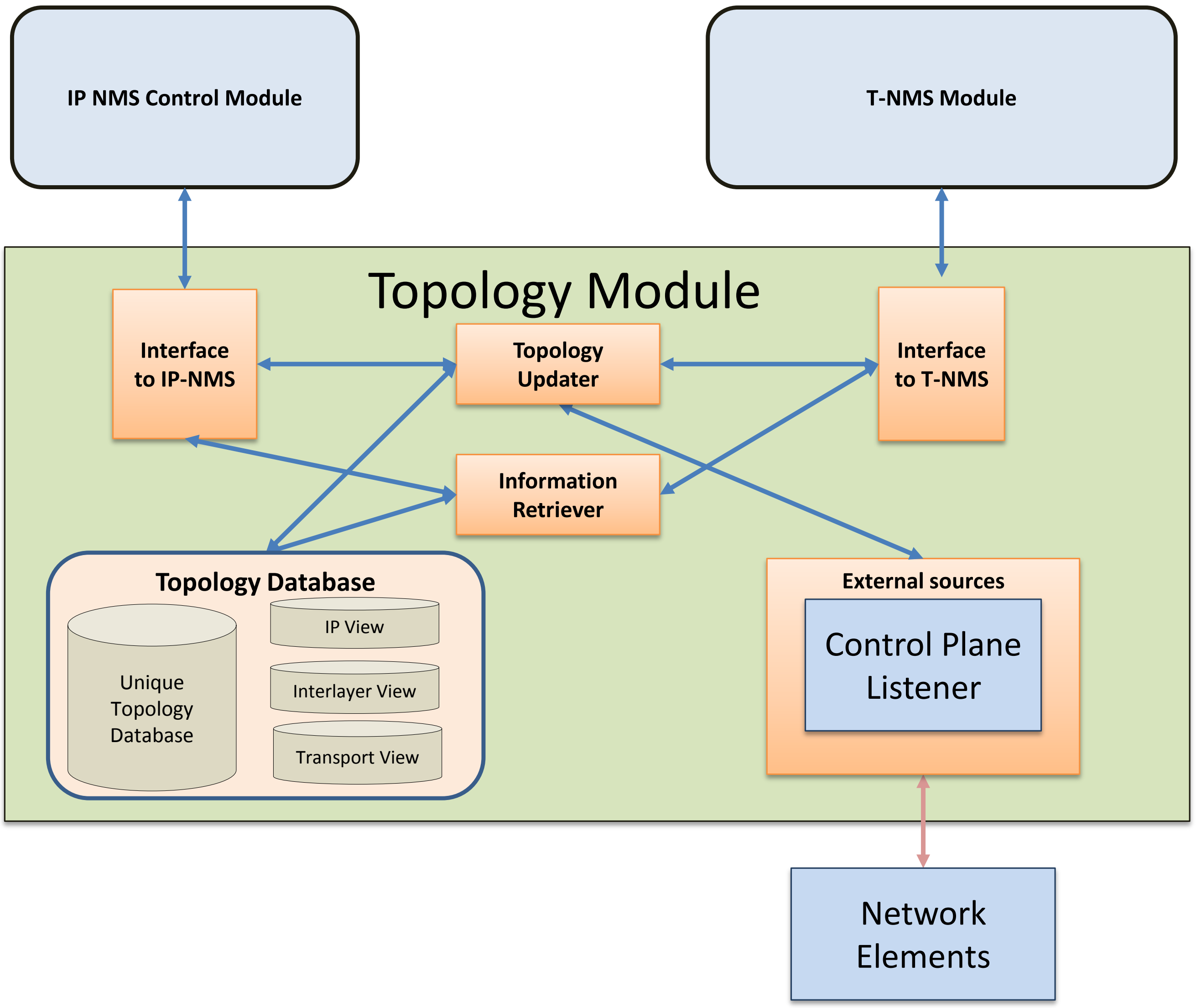


Figure 36. The Topology Module.

One part of the module is devoted to get and maintain up to date the topology from the available sources, which can be the IP-NMS Module, the T-NMS Module or sniffing the control plane information from external sources (e.g. routing protocolos, mainly for the more dynamic information). The topology module maintains the topology by the information sent by the NMSs control modules about topology changes detected in each layer. It is supposed to be used when the ONE adapter encounters topology changes in the network. Other way of maintaining the topology is the possibility of a control plane listener entity for paying attention to topology changes in networks where there is no NMS. It is being discused by the project if that module inclusion is really needed and where it should be placed in order to get the best performance. So it will not be necesarelly included in the final module design.

The other part of the module is devoted to provide the information to the requesting parties. Several funcions have been identified to retreive information:

- GetFullTopology (layer, domain ID): Gets all the topology information of a given layer. In case the layer has several domains, they can be identified by a domainID.
- GetOppositeNode(interface): Retrieves the node in the other layer to which the interfaces is connected.
- GetNeighbourNodesOf (node): Retrieves the neighbour nodes of a given node in a layer.
- GetOppositeInterface(interface): Retrieves the interface in the other layer to which the interface is connected.
- GetNodeByName(node name): Retrieves the node.
- GetIntfByName(interface name): Retrieves the interface.

A more detailed description of the interfaces can be found in Section 12.9.

Topology Module Administration

The administration of the topology module consists of the topology network definition file (in case it is neccesary) and a log file where the information of the execution will be written. The topology file is needed to fed the topology module with a initial topology when it starts. Other possibility, not implemented yet, is topology module feds up from the OSPF messages sent by the network elements.

5.6 Measurement Module

The Measurement module, shown in Figure 37 is responsible for providing the measurement data to requesting module. Data that is being requested from Measurement Module might concern higher level entity within topology e.g. service. In such case Measurement Module needs to identify all entities involved in the request, and poll data from all involved Control Modules. After reception of responses, data has to be combined and formed as an answer.

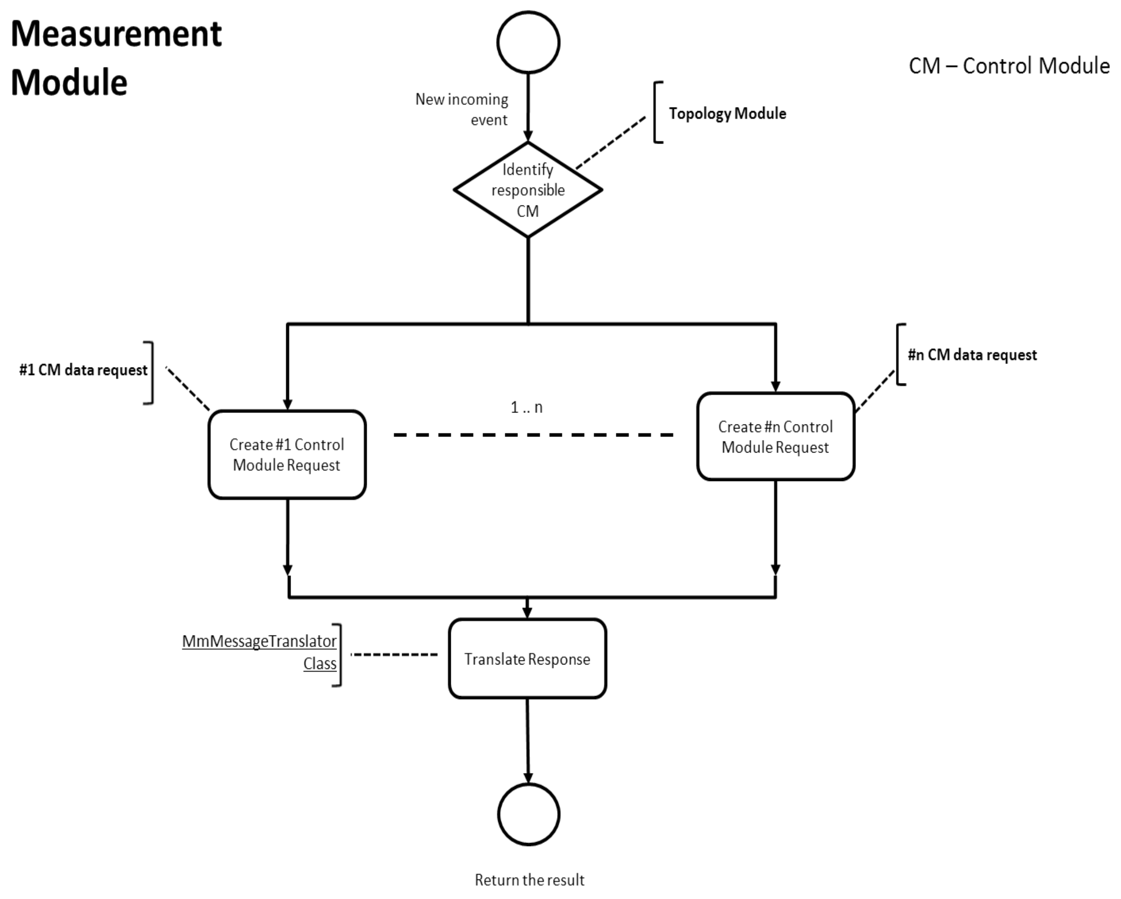


Figure 37. Measurement Module getHistoricalPmData and getCurrentPmData services.

5.7 Programmable Logic Module

The Programmable Logic Module (PLM) provides the interfaces for invoking operations of external (3rd party) systems that can provide specialized logic for management operations in the ONE adapter. One of the modules developed in the ONE adapter in this context is the PCE module which provides an interface to the Path Computation Element (PCE) [PCE01] as shown in Figure 38.

To integrate the PCE within the PLM framework, a PCE-specific session management module is developed that implements a PCE client. The PCE client is used to connect to a PCE server to request path computation, and format the output from the path computation to the ONE-specific path definition. The connection between the server and client can be re-used to serve multiple path computation requests, so the session management function is also responsible for identifying the responses from the PCE server and mapping them to the corresponding requests that were made by the workflow processor. The interface exposed by the PCE PLM module is shown in Section 12.11.1.

The capability of interacting with third-party systems is significant to the applicability of the ONE adapter in current and upcoming management ecosystems, which incorporate a lot of external logic and analytics in determining operations in the network.

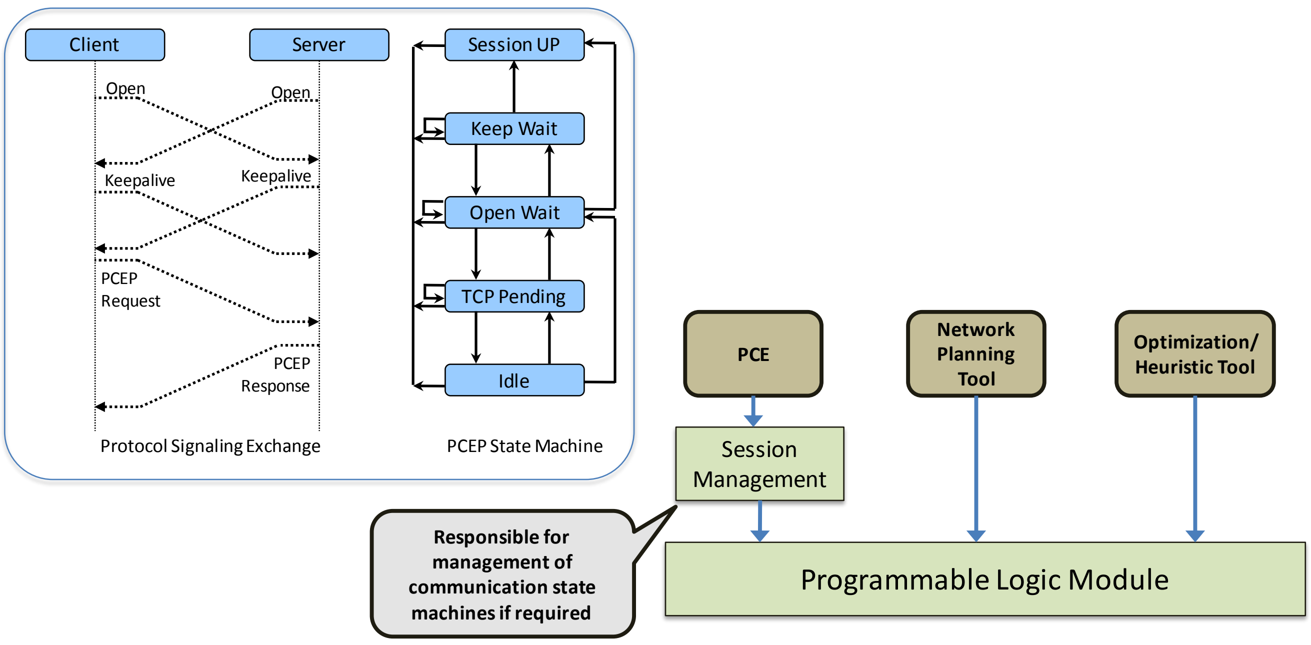


Figure 38 Programmable Logic Module.

5.8 AAA

In this section is described the Authentication, Authorization and Accounting (AAA) module, its management requirements, its functions, and the AAA protocols. Based on the ONE adapter requirements, we propose the specification of the AAA architecture for ONE Adapter.

5.8.1 ONE adapter AAA management requirements

The ONE Adapter AAA module provides authentication, authorization and accounting services. As a result, it might be a potential target for attacks by intruders. That's why the management aspect of AAA is required to protect, log, report, and alert a management entity about any failures. Such failures might be indicative of unlawful intrusion attempt, component failure, or simple user error. To prevent those failures, IETF provides guidelines [RFC 4962] for the AAA management designers, security professionals, and other software developers to predict and protect mistakes and errors when developing new systems and standards.

One of the management requirements that need to be considered when developing the AAA module for the ONE adapter is that the AAA key management protocol must be cryptographic algorithm independent. For example, according to [RFC3579], RADIUS [RFC2865] should be protected by IPSec[RFC2401]. Another requirement is "Strong, fresh session keys", which means, in addition to algorithm independence, the cryptographic key must be strong and frequently changed. And so, the AAA architecture must generate separate and new keys for each session, protecting the system from disclosing the entire system when a successful system attack disclosed one session key. Recommendations of key management issue are provided by National Institute of Standards and Technology [SP800-57].

The ONE Adapter AAA module has the ability to add, delete, edit user, defining the property for each user, add/delete/update policy rules for authorizing administration operations for different modules, and add/delete/update operations on policy for allowing users to trigger workflows.

5.8.2 AAA modules Functions

The AAA module is responsible for protecting the ONE Adapter from attacks that have as target the alteration of the network correct functioning. One of the key functions of the AAA module is the Authentication function, which is the process of validating the identity of a user, or device that requests access to a service or device.
Another key function of the AAA is the Authorization function, which is the process of determining the resources that certain user is permitted to access. A resource could be individual files, devices or access to services. The last key function of the AAA is Accounting which refers to the process of keeping track of

the consumption of network resources by users. The accounting function is used for management, planning, and billing.

One of the main objectives of the ONE Adapter's functional design is to allow the adapter to communicate with external subsystems (MS13). Radius protocol and Diameter protocol are the two main protocols being used in IP and carrier-grade networks for authentication, authorization and accounting purposes. The ONE Adapter AAA module integrates a Radius client, which allows the ONE Adapter to communicate with external AAA providers. This allows unified billing and SLA negotiations with external sub-systems. While the ONE Adapter is capable of directly communicating with external Radius-protocol-based AAA systems, it needs to communicate with Diameter-based-AAA systems via an upgraded path provided by Diameter. It is also possible for the ONE Adapter AAA module to communicate with external Diameter-based AAA modules via a tunnel if Diameter does not provide upgraded path.

5.8.3 AAA Protocols

The following three AAA protocols are mainly used in the network industry:

1. Tacacs+ (Terminal Access Controller Access Control System plus) is the Cisco proprietary [draft-tacacs-02], specifically designed to be fully compatible and scalable with Cisco network devices and servers. Tacacs+ provide Authentication, Authorization and accounting separately via Transmission Control Protocol (TCP)

2. Radius (Remote Authentication in Dial-In User Service) is a standard client/server protocol developed by Livingston Enterprises, Inc. in 1991 and later moved into Internet Engineering Task Force (IETF) standards [JV06]. As opposite to Tacacs+, Radius provides Authentication and Authorization in one process, and is mostly used by ISPs because it uses fewer CPU cycles and less memory. Communication between the client and the server side of Radius is based on the User Datagram Protocol (UDP) [RFC 2865]. Extensions for the Radius protocol, Remote Authentication Dial-In User Service (DIAMETER) are in the process of standardization by the IETF. The main changes in Diameter protocol are the ability to transfer data via TCP and allowing authentication and authorization procedures to be performed separately which in RADIUS has not been implemented yet.

3. PPP (Point-to-Point Protocol) is a data link protocol which provides authentication, transmission encryption, and compression between two network nodes. It has different types of services such as Point-to-Point Protocol over Ethernet (PPPoE) and Point-to-Point Protocol over ATM (PPPoA). It is mainly used by Internet Service Providers (ISPs).

Based on the above mentioned AAA protocol features, as well as based on the requirements of the ONE Adapter, we used Radius protocol for the implementation of the AAA module. It is because Radius protocol fully supports all the ONE Adapter AAA management requirements and, at the same time, it is simple. In addition, Radius protocol is compatible with Diameter protocol, which is also used by many enterprise ISPs.

5.8.4 ONE adapter AAA functional description

The AAA module will support authentication and authorization. The AAA module functional blocks and its interfaces are presented in Figure 39 (refer to Appendix B, Section 12.12, for more on AAA module interface definitions). This figure shows that the AAA module consists of an AAA Engine, a Policy DB and an Application Specific module. The AAA engine is used to receive requests for authentication; authorization and accounting buy checking and storing this information in Policy Database (Policy DB).

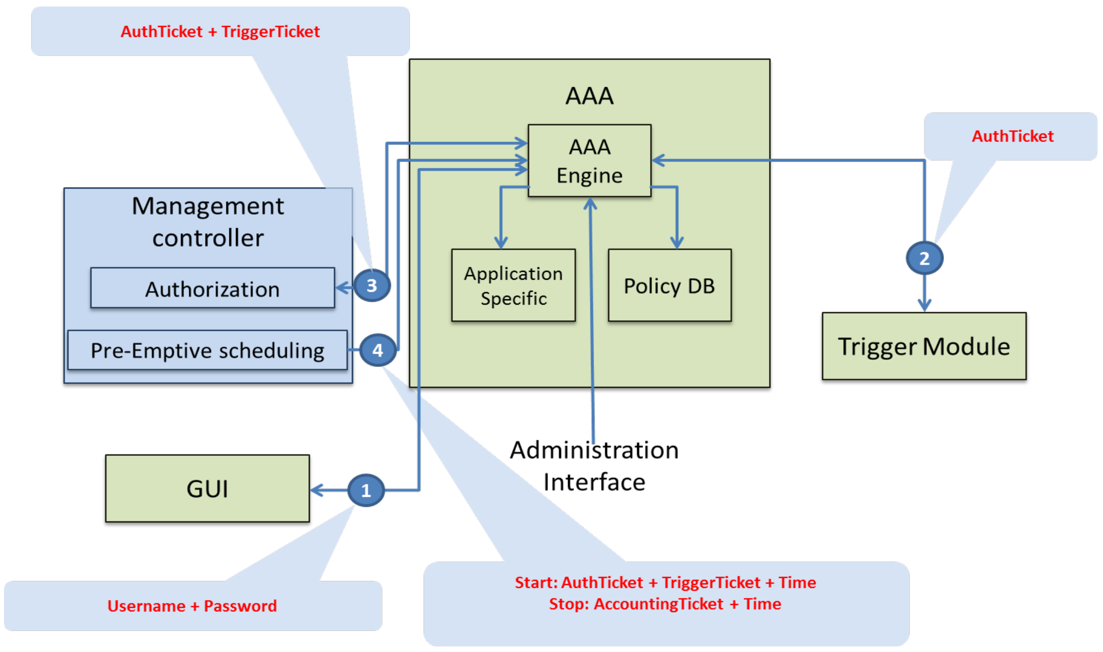


Figure 39 AAA functional modules and interfaces

The interfaces from TM, GUI and smart analytics to the AAA module are used to support initiation of services the ONE Adapter provides to the users. The administration interface on the other hand is used to allow the users to make changes to the access rights (e.g., the update group memberships and group rights) and to the login entries of the AAA DBMS. The administration interface interfaces is also allowing users to access the ONE Adapter core and auxiliary modules to make the necessary changes such as adding, removing, or updating workflows.

5.8.4.1 Authentication Function

First, the user information is sent to the AAA module for authentication either via WSGUI-AAA interface (cf. Section 12.12.1), or admin interface (cf. 12.12.5). Then, the AAA module checks the validity of username and password. If they are correct, it replies back AuthTicket, Adminlevel, and Network
with a Boolean type message "yes" which means the user is a valid person. If the user information does not match with the stored in the database information, it replies a Boolean type message "No" meaning the user is not a valid person and consequently the AAA module denies him access to the services.

5.8.4.2 Authorization function

To grant users right to access the ONE Adapter auxiliary modules and core modules, as well as to grant users the right to initiate the services provided by the ONE Adapter, the user's trigger ticket is sent to the AAA module via the authentication interface. After checking the user's access level, the AAA module replies to the request with a Boolean type message "yes" or "no". For instances, a user request the ONE Adapter services. In this case the TM sends authentication ticket, trigger ticket and trigger information to the WT module for further processing. To check whether the requester has the rights to trigger the requested service, the WT sends a request to the AAA module to check the rights. The AAA module checks the authentication ticket and the demanded service (trigger ticket) with respect to the rights of the authentication ticket and trigger ticket. It replies to the WT with 0 ("Yes"), if the service requested matches the requester's rights, and 1 ("No"), if it does not match or -1("Error").

The AAA also provides an interface for authorizing administrative operations. As described in Section 4.1.5, a user provides authentication credentials (AuthTicket) to every module when performing any administration operation. The module then requests the AAA using the interface (cf. Section 12.12.1) to a) see if the AuthTicket provided belongs to a valid user session and b) verify if the user has sufficient rights to perform the requested administration operation. The response from the operation is an integer with 0 indicating that the user is authorized to perform the requested operation and 1 indicating otherwise.

5.8.4.3 Accounting function

The ONE adapter can also initiate the recording of accounting sessions at the AAA module, facilitating accounting, charging, and billing of multi-layer operations performed by the ONE adapter.

6 Communication with External Systems

The ONE adapter uses various protocols to interact with external systems in the network.
Table 1 outlines some of the possible protocols that will be used in our implementation in WP4 and the modules that would employ them for communication with external systems. The section then provides a brief description of the various protocols indicated in
Table 1.

		EXTERNAL ACTORS.				
		IP-NMS	T-NMS	AAA	OpenFlow	PCE
ONE MODULES	IP NMS	**SNMP/NETCONF/CLI:** Request/Set configurations, state, statistics.	X	X	??	X
	T-NMS	X	**MTOSI:** Request/Set configurations, state, statistics.	X	X	X
	AAA	X	X	**RADIUS/DIAMETER:** Request device or user authentication.	X	X
	Trigger	Receive network alarms (SOAP based)	Receive network alarms (SOAP based)	X	x	X
	Path Computation Client	X	X	X	**x**	**PCEP** Request a Path computation

Table 1.One external Communications.

6.1 PCE Communication Protocol (PCEP)

The Path Computation Client is an instance of a Programmable Logic Module, which may be used for computing multi-layer paths in the network. The Path Computation Client Module will interact with an external Path Computation Element (PCE), which is a centralized server used to compute paths.

A PCE is a node that has specialized path computation capabilities and receives path computation requests from entities or clients called Path Computation Clients (PCCs). The use of a PCE eliminates the need for path computation capabilities in every node within the network. For instance, there is no need for every node in the network to maintain a path computation database; there is now a central database, which can be used for path computation purposes. In order integrate the PCE with the ONE adapter, the communication between the ONE adapter and the PCE can be accomplished using the PCEP protocol.

One of the auxiliary modules of ONE adapter acts as a PCC enabling access to the features and strengths offered by a PCE. The interface for path computation request can be found in the Appendix B, Section 12.11.1, with all relevant parameter descriptions.

6.2 RADIUS/DIAMETER

In order to insure the ONE Adapter interoperability with external AAA providers, the ONE Adapter AAA module is implemented on the bases of the RADIUS protocol (For more information, refer to Section 5.8.3). The RADIUS and DIAMETER are the two main AAA protocols being used in telecommunication network. The RADIUS protocol was designed to provide a simple but efficient way to deliver AAA capability. The DIAMETER protocol was derived from RADIUS, and designed to address new requirements that arose from the evolution of network applications and protocols. While the ONE Adapter can directly communicate with the RADIUS protocol based external AAA providers, the DIAMETER protocol, though not directly backward compatible to the RADIUS protocol, provides an upgrade path for RADIUS.

6.3 MTOSI

Multi-Technology Operations System Interface (MTOSI) is a protocol developed by Telemanagement Forum in order to standardize operation between different Operational Support Systems (OSS). It is defined to support network resources management and operations, as well as the services management and operations. MTOSI is defined to support various technologies in the Optical and Ethernet areas, therefore is being used to interconnect EMS (Element Management System) layer with NMS (Network Management System) layer with support of Multi-Technology Network Management (MTNM) standards, and also NMS layer with OSS apllications.

MTOSI is based on web technologies such as JMS, XML, Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL).

Core functionalities of the management system are supported by it natively. Within the main we can enumerate: Inventory retrieval interface for the configuration service, Device discovery and provisioning interface, Alarm monitoring interface with support for coarse-grained alarm retrieval operations for Fault Service, Topic based publisher and subscriber notifications for the Notification Service, Service Activation interface, and Billing interface etc.

6.4 OpenFlow

OpenFlow is a technology that allows control and management of programmable networks. It was primarily designed for Layer 2 networks; however it is applicable for any other if the right agent has been developed. It resolves problems of creation network topologies, bypassing limitations of SNMP and vendor specific protocols.

Architecture is made of agents and its controllers. Agents operate at lowest level and their protocol is binary. Agents are connected through OpenFlow controllers. Controllers may be extended by any software defined modules that can give and outside interface for controlling network or even self-maintain its nodes.

OpenFlow network model is simplified as much as possible. Switch is the most generic element and may contain ports. Ports have its attributes including basic performance monitoring data like number of Rx/Tx packets. All available counters are described in OpenFlow Switch Specification [OpenFlow].

Every switch may have flows that are set of rules similar to ones used in firewalls which can utilize on

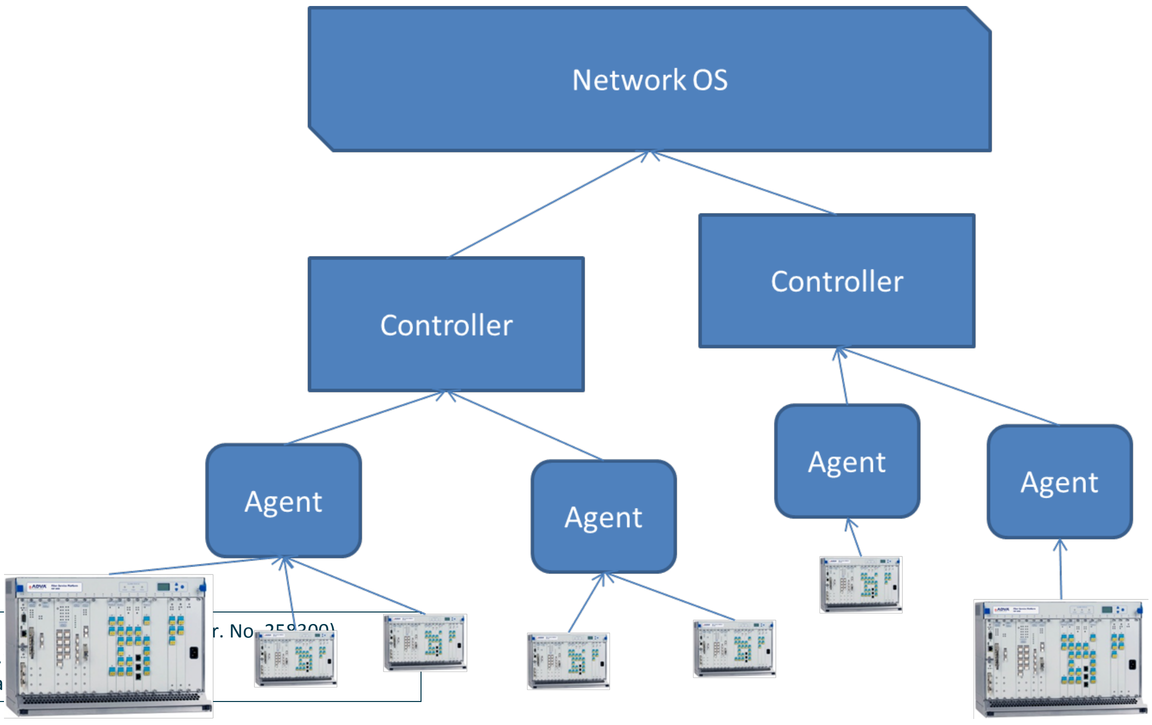


Figure 40 OpenFlow deployment

of following actions:

- Forward
- Encapsulate and forward
- Drop
- Modify-Field

Communication with OpenFlow is done through REST Web Services served by Floodlight controller.

6.4.1 Rest interfaces used in Communication with Floodlight OpenFlow Controller

Communication with Floodlight OpenFlow is done by REST API. Any arguments are passed though JSON format described in RFC 4627. Due to ONE requirement controller uses functions that allow topology information retrieval, services management and path routing.

Floodlight also allows to be extended with user developed modules in its API that gives good opportunities in feature for probable project extension.

6.4.1.1 Static Flow Pusher

Static Flow pusher allows flows management. Flows are abstract connections that mean different things for different agents. ADVA VSwitch Agent will map them into Control Plane services. In this configuration there is only one switch that contains ports that represent ECH from different nodes. When flow is created between them, Control Plane does tunnel which is a service in its point of view.

URI	**Arguments**
/wm/staticflowentrypusher/json	HTTP POST data (add flow), HTTP DELETE (for deletion)
/wm/staticflowentrypusher/list/<switch>/json	**switch**: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) or "all"
/wm/staticflowentrypusher/clear/<switch>/json	**switch**: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) or "all"

6.4.1.2 Switch Data Retrieval

This command gives information about topology structure. In Big Switch configuration it is mapped into node in Topology Module. Ports are interpreted into interfaces, this translation is correct according to ONE's topology model.

Ports are endpoints from every managed node. This is low detailed information but sufficient for ONE, because one of its goals is to allow communication between hardware manufactured by different vendors where high abstraction is an advantage.

Every port contains of performance monitoring data for TX/RX data packets and bytes that provide most basic network optimization information.

URI	Arguments
/wm/core/switch/all/<statType>/json	**statType**: port, queue, flow, aggregate, desc, table, features
/wm/core/switch/<switchId>/<statType>/json	**switchId**: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) **statType**: port, queue, flow, aggregate, desc, table, features

6.4.1.3 Routing

Controller can find shortest route for their services. It has no use in Big Switch configuration but can be utilized in classic applications for Ethernet networks where topology can be achieved by built in functionality.

URI	Arguments
/wm/topology/route/<switchIdA>/<portA>/<switchIdB>/<portB>/json	**switchIdA**: host A connected switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) **portA**: host A connected port on switch **switchIdB**: host B connected switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) **portB**: host B connected port on switch

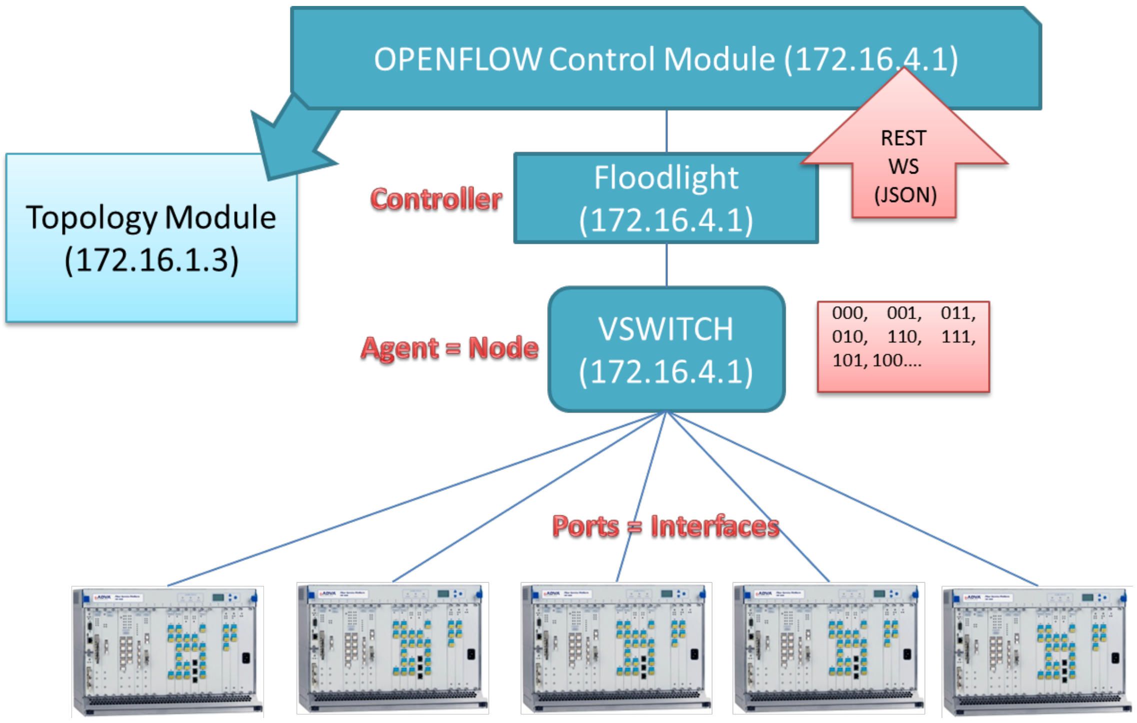


Figure 41 Illustration of specific implementation for Openflow implementation with One Big Switch configuration and illustration of mapping them into topology.

6.5 SNMP

SNMP is a standard protocol available for nearly all network devices and its standard MIBS are implemented in all these devices. The standard MIBS permits monitoring the state of the links by polling thorough SNMP get messages, or asynchronously using SNMP traps. This protocol is very useful to identify the vendor, and software version of the NE, enabling one approach or another at the time to manage the device by the ONE adapter

This protocol is supported by the ONE adapter because interoperability reasons and it ease of use in nearly all networks devices to provide information to several modules of ONE, like the IP-NMS and the trigger module, on network usage and failures.

On the other hand this protocol is inadequate at the time of configuring network devices, because while the message transport format for SNMP set is standard, the MIBS are not, so this require the installation of proprietary MIBS; but the worst problem is that the vendors that supports a full SNMP equipment configuration is rare, so this protocol is good for monitoring but unfeasible for NE configuration.

6.6 NETCONF

The network configuration protocol is an IETF standard defined in RFC 6241, it has the capability to manage and configure networks elements through a simple RPC (Remote Procedure Call) mechanism. The messages are interchanged are XML encoded.

While the message types are well defined, the management and the configuration details are vendor proprietary and frequently is an XML translation of the vendor's CLI syntax so an adaptation is mandatory in all cases.

This protocol is mandatory in the ONE adapter because it is a standard, yet incomplete but some vendors supports it and over the time this will be the preferred configuration protocol.

6.7 CLI

This is not a protocol itself, but all network devices haves one configuration console where it is possible to do a full configuration of the device and its services. This console requires a telnet or an ssh client. The configuration via CLI is proprietary in all the cases and not efficient, giving to loosing packets in multilayer reconfigurations in most of the situations, but works with all vendors, so is the solution when no any other protocol is available.

7 Orchestration addressing and use cases

In the context of the ONE Adapter, we have developed four use cases, namely: IP Link Provisioning, IP Service Provisioning, IP Offloading and Post-Failure recovery. This section provides a detailed view of the interactions and communications between modules for each of these use cases.

7.1 IP Link Provisioning

The IP Link Provisioning use case refers to the automated operational coordination and inter-layer communication between the IP and Optical Management ecosystems through the ONE Adapter, for setting a link between a pair of routers in the IP Network. It involves several intermediate operations, including:

- Check for virtual/physical availability of interfaces at the IP Layer for the routers involved in the provisioning scenario.
- Identification of corresponding transport network switches; this refers to the IP-Optical Interface correlation.
- Transport Network circuit reservation between optical end-points.
- Data plane configuration for circuit end-points.
- IP interface configuration, including IP addresses and routing rules.

Next, we'll provide a detailed description of the flow of actions performed by the ONE adapter in order to automatically provide an IP link in a coordinated fashion. This flow description will allow the identification of internal and external actions and communications between the ONE Adapter's modules. Figure 42, Figure 43, Figure 44 and depict the general operational flow for the IP link provisioning use case. A total of twenty-three steps complete the sequence of actions between modules. We assume that the IP link provisioning workflow has been triggered by a Network Operator through the Graphical User Interface (GUI).

Next, we'll describe on a per-step basis the sequence of operations.

1. **GUI – AAA**: The GUI must interface the AAA module for user authentication. For this purpose, the GUI sends to the AAA a user-name and the password parameters.
2. **AAA – GUI:** In response to the authentication request, the AAA module will retrieve a ticket number (Authorization Ticket).

3. **GUI – Trigger Module**: Once the user has been authenticated, the GUI initializes the link provisioning request. Sending out to the trigger module mandatory and optional configuration parameters required for performing such operation plus the granted authorization ticket. These parameters are the Event ID, the AuthTicket and Params option. The Params option contains all the options necessary for provisioning an IP link (as it was described in Section **Error! Reference source not found.**. For more information regarding the communication between the GUI and the Trigger Module see Section 12.5.1.
4. **Trigger Module – Management Controller**: Processes and analyzes a trigger in order to execute a workflow. For this purpose the method MCAddWorkflow is invoked and the parameters involved in this method are: authTicket, triggerTicket, triggerID, triggerParams, log.
5. **Management Controller – AAA**: requests for trigger authorization. For this purpose the Triggerticket and the AuthTicket parameters are sent.
6. **Management Controller – Workflow Processor**: upon authorization it retrieves the workflow ID.
7. **Workflow Processor**: retrieve the instance of workflow from the *workflow database* for execution. For this purpose the method startWorkFlowExcution is called. This method receives as inputs the following parameters: TriggerTicket, AuthTicket, parameters and log.

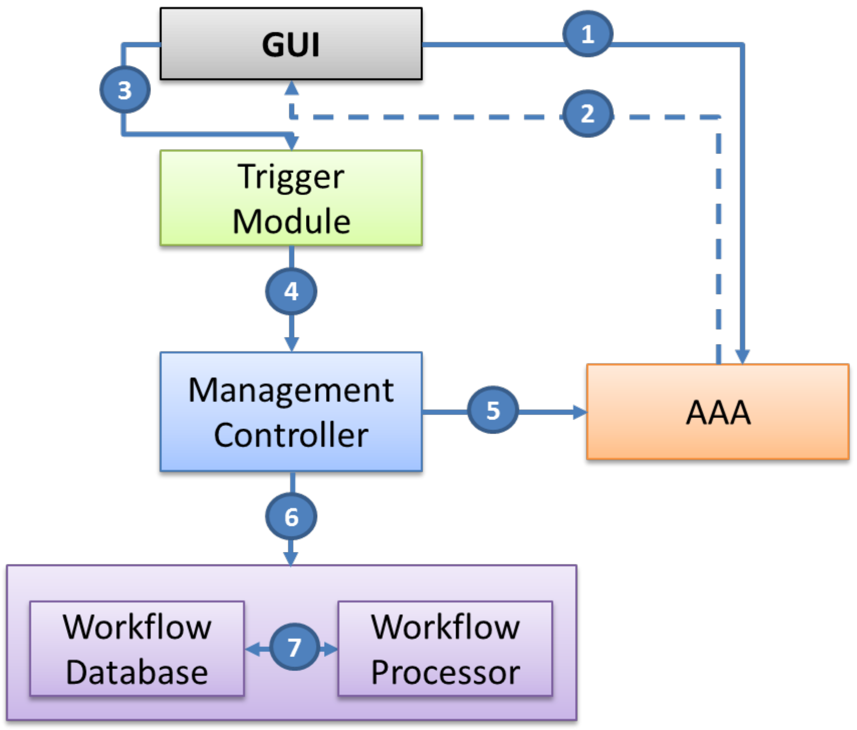


Figure 42.Operational Flow for IP Link Provisioning Use Case – Part I.

8. **Workflow Processor- Topology Module**: Request photonic Interface given the pair of IP end-points. For this purpose the method GetOppositeInterface is invoked and the parameters involved during the invocation of this process are. intfName, Address, Layer, IsPhysical, ParentIntfName. For more information see Section 4.3.
9. **Topology Module – Workflow Processor**: Retrieve Photonic Interface to the Workflow Processor.
10. **Workflow Processor – T-NMS Module**: Request provisioning of optical circuit given the two optical end-points (photonic interfaces).
11. **T-NMS Module – Ontology Mapper**: request syntactic adaptation to the Ontology Mapper for external communication with Transport's Network Management System.
12. **Ontology Mapper: T-NMS Module**: retrieve configuration statements for optical circuit provisioning.
13. **T-NMS Module – External T-NMS**: Send optical circuit request directly to external Network Management System via MTOSI.
14. **External T-NMS – T-NMS Module**: retrieve setting acknowledgement.

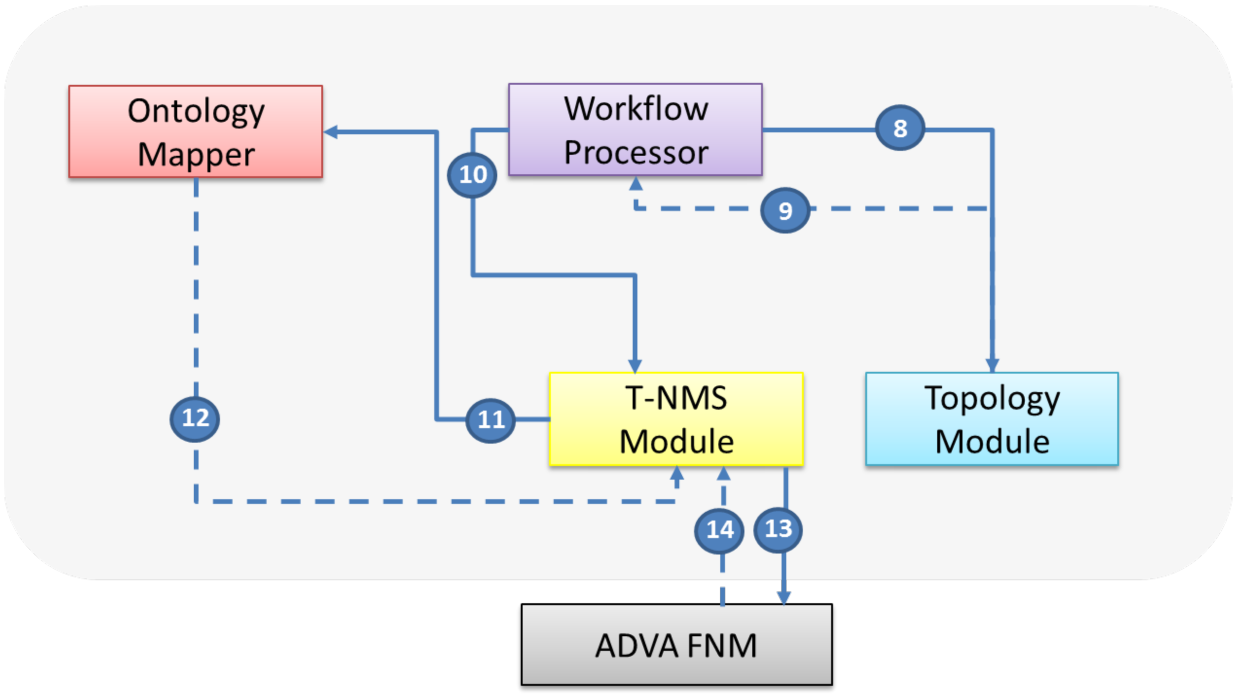


Figure 43 Operational Flow for IP Link Provisioning Use Case – Part II.

15. **T-NMS Module – Workflow Processor**: Notify successful optical circuit provisioning
16. **Workflow Processor – IP-NMS Module**: request configuration of IP Link.
17. **IP-NMS Control Module – Ontology Mapper**: Request syntactic adaptation to the Ontology Mapper for external communication with IP's Network Management System. The IP-NMS Control module communicates with the Ontology Mapper using a JSON String containing the following parameters the following parameters: routerVendor, routerModel, routerIOsVersion, action, confMode, optParams, routerVendor, routerModel, routerIOsVersion, action, confMode, optParams.

18. **Ontology Mapper – IP-NMS Control Module**: Retrieve configuration statements for configuring routers at the IP layer. For this purpose the Ontology Mapper sends a JSON String to the IP-NMS Control Module, which contains the following parameters: module, sentence.
19. **IP-NMS Control Module – External IP-NMS**: send configuration statements to external network management systems. In the case of Magalia through the use of Web Services. For

the case of Magalia, the methods configureIPLink, configureUNIRequest and configureIPStaticRoute are called. For more information regarding the parameters involve during the invocation of these methods see the section IP-NMS Control Module of the Section 12.6.

20. **External IP-NMS – IP-NMS Control Module**: Retrieve configuration acknowledgement.
21. **IP-NMS Control Module – Workflow Processor**: Notify successful router configurations.
22. **Workflow Processor – Management Controller**: Notify end of Workflow Execution. For this purpose the method MCSetWorkflowStatus is invoked using the following parameters: triggerTicket, statusCode, statusMessage, log.
23. **GUI – Management Controller**: UpdateGUI status.. For this purpose the Management Controller sends status-information to the Notification module. The GUI can access the status-information store in Notification Module by polling strategies

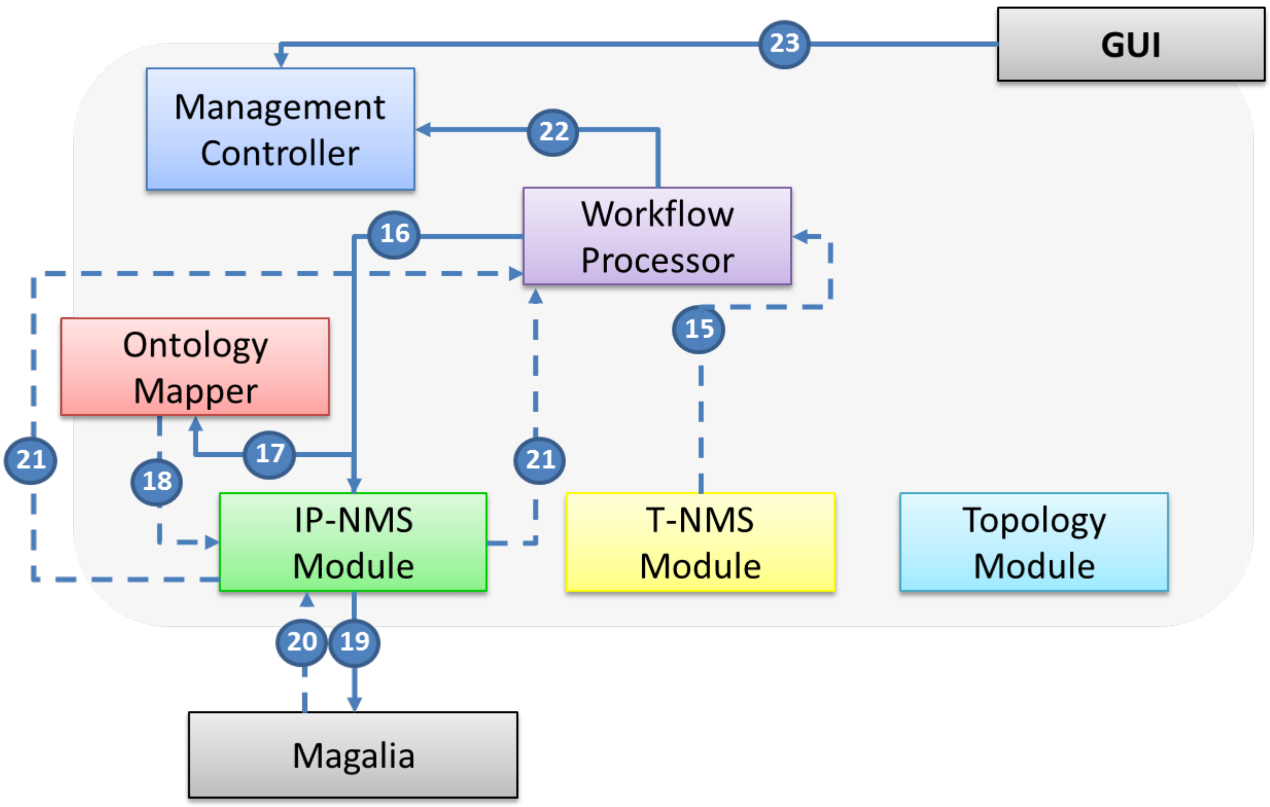


Figure 44 Operational Flow for IP Link Provisioning Use Case – Part III.

7.2 IP Service Provisioning

The IP service provisioning use case is related to the ability of the ONE adapter for provisioning IP services in a network, such as a VPN or an IPTV service. The set of different IP services that can be provisioned in a network is extensive and heterogeneous, i.e., each IP service requires different configuration actions. As a consequence, it is impossible to have a workflow definition that covers all the possible set of IP services. However, all the cases of IP service provisioning are formed by several atomic actions, such as IP link provisioning.

Therefore, the internal operation of the ONE adapter and the external actors the ONE adapter communicates with, are frequently the same. Nevertheless, the input parameters required for the execution of a workflow are different according to the type of service to be provisioned. For instance, for provisioning a VPN service, independently of the type of VPN desired (encryption protocol employed, etc.), there is a set of input parameters that are mandatory, such as the tunnel source and destination address. However, additional parameters may be needed in order to accomplish the desire VPN service.

Moreover, it is important to remark that on the contrary to an IP Link Provisioning use case in which two IP endpoints are being configured; in the IP Service Provisioning use case more than two endpoints could be configured.

This deliverable does not show the internal and external communications of the ONE adapter for an IP service provisioning, because as mentioned above, the input parameters provided from the event listeners is what differentiates an IP service from another.

7.3 IP Offloading

The IP application offloading use case (presented in Deliverable D2.1 [D.2.1]), describes a scenario where application traffic is offloaded end-to-end across the core network on a dedicated circuit.

As presented in previous documents of this project, the IP Offloading case is just a particular, policy driven and automated case of the IP link/service provisioning one. The IP Offloading case consists in the ONE Adapter to monitor network links and detect traffic growth over a previously defined threshold. Then, the ONE Adapter will have to, based on operator defined policies or decision, compute the optimum path (across both IP/MPLS and transport layers) to offload the traffic creating a new IP/MPLS link over the transport network.
As this use case ends with the IP link provisioning case, the orchestration shown will end with the call to the Management Controller Module with a properly defined trigger to initiate a link provisioning (or more than one) workflow.

Initiation action

The IP Offloading workflow could be initiated by the network operator or as the result of an incoming network alarm (a web service call sent by the IP-NMS). In the case it was initiated by an network alarm, a correlation action is need it in order to know where does the traffic go and, according to that, the ONE adapter is going to elaborate an adequate trigger done by the trigger module.

Workflow Execution

In case that the IP Offloading was initiated by the operator, the execution steps are the same as in the IP Link case. In case that the workflow is initiated automatically by a network alarm, the steps will change.

The workflow steps are the following (also they are shown in Figure 45)

1. **Trigger Module – AAA**: The Trigger must interface the AAA module for user authentication for the network alarm sender.
2. **AAA – Trigger Module:** Authorization Ticket sent by the AAA module.
3. **Trigger Module**: The trigger module correlates management information in the past to be sure about what's the reason of the trap and if an IP Offloading should be performed. Assuming a yes for this use case, it will compose a TriggerTicket by looking up the corresponding TriggerID for the incoming EventID and attach a timestamp to make the TriggerTicket unique.
4. **Trigger Module – Management Controller**: The TriggerTicket, the AuthenticationTicket and the optional JSON Event parameter are passed to the Management Controller for further processing.
5. **Management Controller – Workflow Database**: The authorization of the TriggerTicket is checked with the AuthorizationTicketinsie the Management Controller. If successful, the TriggerTicket is used to retrieve the corresponding WorkflowID and Priority in the Workflow Database.
6. **Workflow Processor**: With the workflow obtained, it is started to be executed.
7. **Workflow Processor – Topology Module:** The workflow processor creates the request for the topology module to get the endpoint information needed to accomplish the configuration of the new IP Link.
8. **Topology Module – Workflow Processor:** The answer with the endpoints.

9. **Workflow Processor:** The IP Link provisioning workflow is started as request from the Workflow Processor

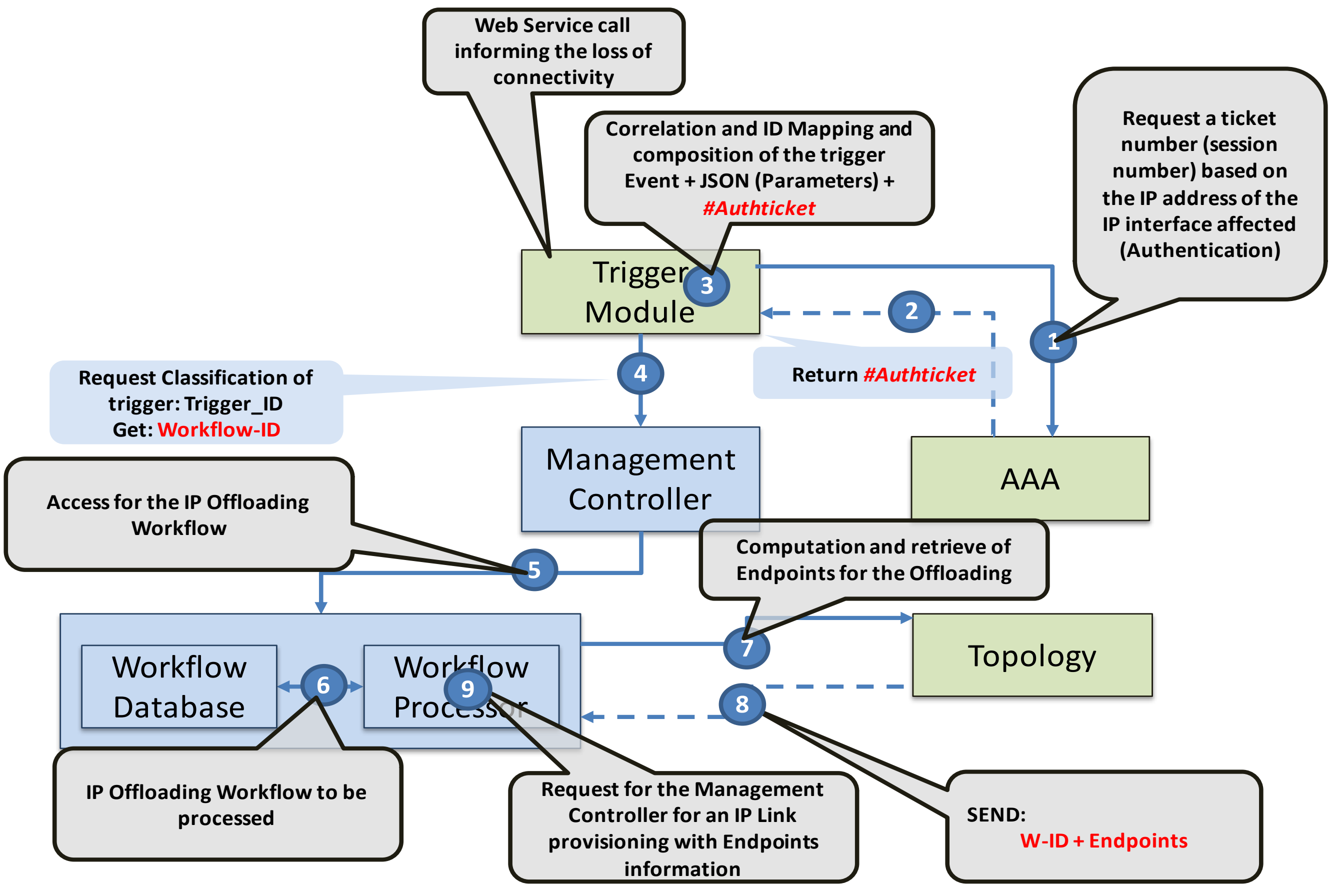

Web Service call informing the loss of connectivity
Correlation and ID Mapping and composition of the trigger Event + JSON (Parameters) + #Authticket
Request a ticket number (session number) based on the IP address of the IP interface affected (Authentication)
Trigger Module
3
2
1
Return #Authticket
4
Request Classification of trigger: Trigger_ID Get: Workflow-ID
Management Controller
AAA
Access for the IP Offloading Workflow
5
Computation and retrieve of Endpoints for the Offloading
Workflow Database
6
Workflow Processor
9
7
Topology
8
IP Offloading Workflow to be processed
Request for the Management Controller for an IP Link provisioning with Endpoints information
SEND: W-ID + Endpoints

Figure 45 Workflow steps IP-Offloading.

7.4 Post-failure recovery

The post-failure case (presented in Deliverable D2.1 [D.2.1]), describes a scenario where the traffic end-to-end is restored in a new path because of a segment network failure.

The post-failure recovery case is just a particular, as Offloading case, a policy driven and automated case of the IP link/service provisioning. The Post-failure case consists in the connectivity restoration when the ONE adapter detects a failing element like a link or NE because the reception of an alarm from the IP-NMS or the T-NMS. The ONE Adapter will create a new path replacing the previous one failed (across both IP/MPLS and transport layers) creating a new IP/MPLS link over the transport network, and deleting the previous one.

This use case ends with the IP link provisioning case, the orchestration shown will end with the call to the Management Controller Module with a properly defined trigger to initiate one or more workflows for:

- Link provisioning the previous link.
- Reroute the traffic through the new paths
- Delete the old paths

Initiation action

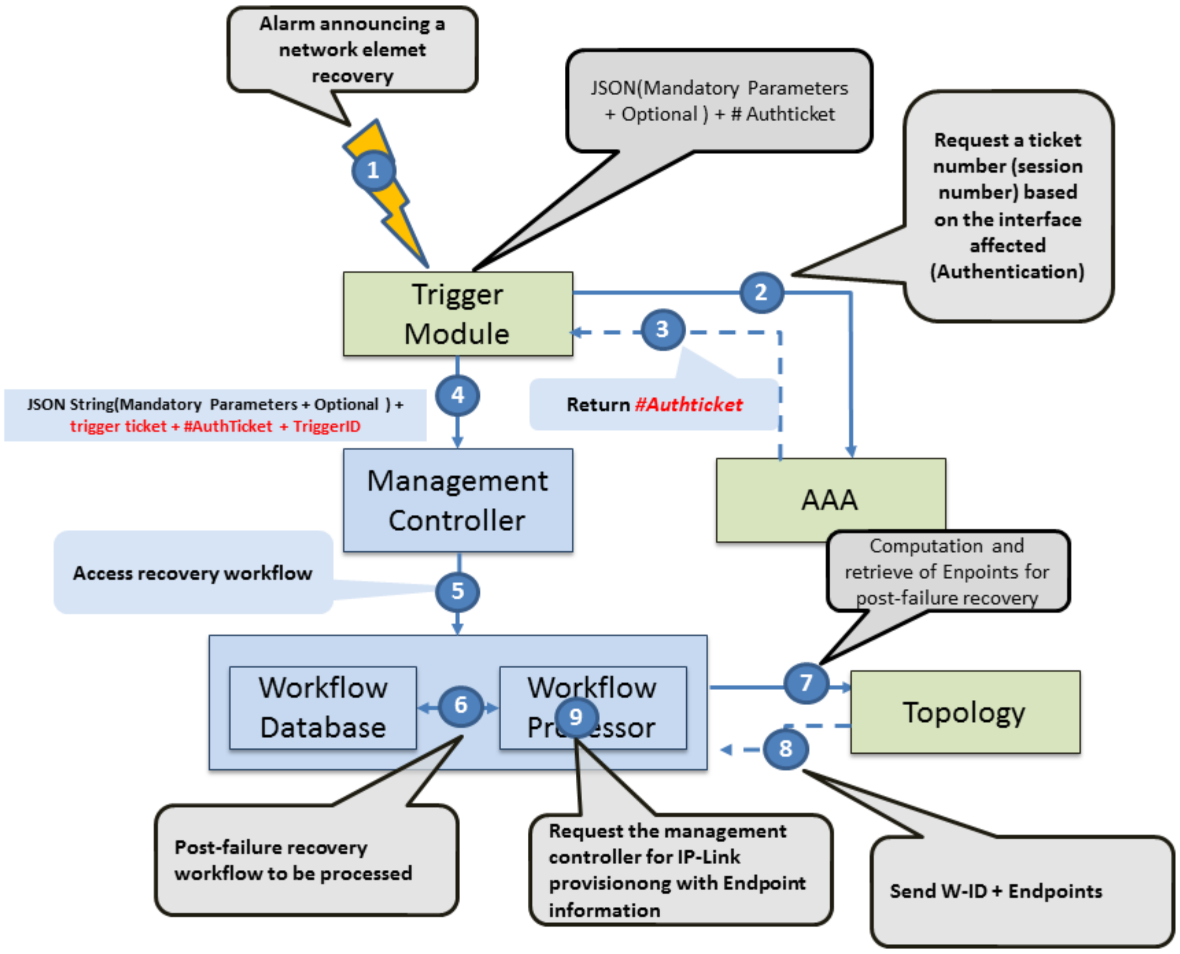
The Post-failure recovery workflow could be initiated by the result of an incoming network recovery event (a web service call sent by the IP-NMS or the T-NMS), a correlation action is need it in order to know where does the traffic go and, according to that, the ONE adapter is going to elaborate an adequate trigger done by the trigger module.

Workflow Execution

The workflow steps are the following (also they are shown in Figure **46**)

1. Alarm announcing a link failure
2. **Trigger Module – AAA**: The Trigger must interface the AAA module for user authentication for the network alarm sender.
3. **AAA – Trigger Module:** Authorization Ticket sent by the AAA module.
4. **Trigger Module**: Compose a TriggerTicket by looking up the corresponding TriggerID for the incoming EventID and attach a timestamp to make the TriggerTicket unique.
5. **Trigger Module – Management Controller**: The TriggerTicket, the AuthenticationTicket and the optional JSON Event parameter are passed to the Management Controller for further processing.
6. **Management Controller – Workflow Database**: The authorization of the TriggerTicket is checked with the AuthorizationTicketinsie the Management Controller. If successful, the TriggerTicket is used to retrieve the corresponding WorkflowID and Priority in the Workflow Database.
7. **Workflow Processor**: With the workflow obtained, it is started to be executed.

8. **Workflow Processor – Topology Module:** The workflow processor creates the request for the topology module to get the endpoint information needed to accomplish the configuration of the new IP Link.
9. **Workflow Processor:** The IP Link provisioning workflow is started as request from the Workflow Processor



Alarm announcing a network elemet recovery
JSON(Mandatory Parameters + Optional) + # Authticket
Request a ticket number (session number) based on the interface affected (Authentication)
1
2
3
Trigger Module
Return #Authticket
4
JSON String(Mandatory Parameters + Optional) + trigger ticket + #AuthTicket + TriggerID
Management Controller
AAA
Computation and retrieve of Enpoints for post-failure recovery
Access recovery workflow
5
Workflow Database
6
Workflow Processor
9
7
8
Topology
Post-failure recovery workflow to be processed
Request the management controller for IP-Link provisionong with Endpoint information
Send W-ID + Endpoints

Figure 46 Workflow steps Post-failure recovery

8 Conclusions

In this deliverable, we have provided the detailed functional design and interface specification of all the modules specified in the ONE adapter architecture. The document describes the modules and the interfaces in detail, describes how operations can be initiated, and provides description of the execution of the ONE adapter use cases based on the presented design. The document also discusses the administration functions exposed by each module that allow the operator to configure and tweak ONE adapter operation.

Some aspects of the functional design has been updated from the previous iterations of this document based on the implementation experiences in WP4, and the final functional design presented here will provide the foundation for the implementation specification for WP4.

9 References

[BPEL01]	M. B. Juric, “Business Process Execution Language for Web Services”, Second Edition Packt Publishing UK. ISBN 1-904811-81-7. 2006)
[D.2.1]	Deliverable 2.2.1 Preliminary report on architectural design of the management adapter
[DRAFT-TACACS-02]	D. Carrel, L. Grant, “The TACACS+ Protocol”, 1997
[JSON01]	Java Script Object Notation, http://www.json.org/
[JV06]	J. Vollbrecht, "The Beginnings and History of RADIUS” Interlink Networks, Cisco AAA Implementation Case Study, 2006.
[MVEL01]	MVEL Expression Language for Java, http://mvel.codehaus.org/
[OPENER12]	OPENER www.craax.upc.edu/opener.html (Last revised on June 2012)
[RFC 4962]	R. Housley, et. al,” Guidance for Authentication, Authorization, and Accounting (AAA) Key Management”, 2007
[RFC 3579]	B. Aboba, “RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)”, 2003.
[RFC 2865]	C. Rigneyetet Al, “Remote Authentication Dial In User Service (RADIUS)”, 2000.
[RFC 2401]	S. Kent, “Security Architecture for the Internet Protocol”, 1998,
[SP800-57]	E. Barker et. Al, ”Recommendation for key management”.
[RFC 3588]	P. Calhoun et AL, “Diameter Base Protocol”, RFC 3588,2003

[RFC 1661]	W. Simpson, “The Point-to-Point Protocol (PPP)”,RFC 1661, 1994.
[PCE01]	A. Farrel, J. P. Vasseur, J. Ash, “A Path Computation Element-Based Architecture”, IETF RFC 4655, August 2006, http://tools.ietf.org/rfc/ rfc4655.txt
[PCE02]	JP. Vasseur, J.L. Le Roux,“Path Computation Element Communication Protocol”,IETF RFC 5440, March 2009, http://tools.ietf.org/rfc/rfc5440. txt
[VLR09]	JP. Vasseur, and JL. Le Roux, "Path Computation Element (PCE) Communication Pro-tocol (PCEP)", RFC 5440, March 2009.
[OpenFlow]	http://www.openflow.org/wp/documents/

10 Acronyms

[AAA]	Authentication, Authorization and Accounting
[AT]	Authorization Ticket
[BPEL]	Business Process Execution Language
[DB]	Database
[GUI]	Graphical User Interface
[MC]	Management Controller
[NMS]	Network Management System
[TM]	Trigger Module
[TP]	Trigger Parameter
[TT]	Trigger Ticket
[IPA]	IP Address
[JSON]	JavaScript Object Notation
[OM]	Ontology Mapper
[OS]	Operating System
[DBMS]	Database Management System
[API]	Application Programming Interface
[GTK/GDK]	GIMP Toolkit/GIMP Drawing Kit
[LM]	Logging and Monitoring
[WFT]	Workflow Triggering
[WFP]	Workflow Processor

11 Appendix A: Configurable Helps

In this section it is briefly detail the work done on Quagga routing suite in order to provide new functions and capabilities, the likes of which were demanded during its development. For now, all the new functionalities work under the CLI (remote console command in Quagga's case), although they might be converted to be called under other interfaces, as seen on other related projects.

Specification

The new functions aimed for two main purposes. The first one was to empower a user to include and associate arbitrary data to a command, for instance several XML tags to be linked as *meta information* of a given command.

The other one, as important as the previous one, was to allow the user to change a command's help description to include (append or truncate) new information as that command's help description.

As the full Quagga suite runs on memory, and the only persistent data is the configuration file/s; we came to the conclusion that we also needed to make the new data models persistent, so that the user might be able to save the data if the Quagga service/daemon was to be restarted, for example. For that, a very simple parser was implemented, and will be described at the end of this report.

Considerations

One of the first thoughts that should be considered upon the given implementation is that, this first delivery tried to focus on solving the problems with simple design and using most of the already-done work inside Quagga.

So to explain, for example we got into the problem where the internal compiler of commands of Quagga is only able to read an arbitrary and single string per command, so if we wanted to read two different string in one command, we should have changed the internal compiler of commands, but instead, to board the problem in less time we decided to divide the work into two commands instead of one. An example of this will be shown onward.

Another thing to be considered is that, for now all data included is being saved during execution time on the Quagga daemon, which for now runs on a computer with its own resources. The fact that we allow writing new data and storing it, is in fact memory usage which we will hardly care about since we have plenty of space of volatile memory to waste on our test machines, but this matter should be looked upon carefully in case of an actual router.

The last consideration we would like to add is that we strongly recommended testing the code and functions deeply since some of them have only been tested in prepared environments, and there might be extreme cases that might have not been thought of during the implementation stage.

Detailed usage of new functions

All new CLI commands are to be executed in *enabled* state. This is part of the Quagga running behavior, thus we won't bring specific details in this report.

Metainformation-related functions

The following part describes every single CLI function that was included to manage the commands that handle the metainformation.

Metainfo commands

```
Router# metainfo
append clear create get set
```

We will now proceed to give detailed information of every command for the *metainfo* branch of commands.

- **# metainfo create** <Arbitrary-String> <String> : Should be the data we want to store or append as a command's metainfo. - The *create* command manages and reserves the space in memory to store the String of metainformation. - By itself is harmless, can handle any size of String of data. - **Only the last created string is maintained in memory for further operations**; imagine that you have a global variable that you can use anytime, and only maintains the content you last entered by the means of the *create* command.
- **# metainfo set** <Arbitrary-Command> <Arbitrary-Command> : Should be a **valid** command we want to have the information embed to**. - The metainformation value is taken from what was last inputted with *create***. If no *create* command was inputted, it just uses an empty String. -This command overrides whatever metainformation was already embed on the command, and cannot be undone. -This command has integrity check so that the inputted command is actually a valid Quagga command, or at least part of it.
- **# metainfo append** <Arbitrary-Command> <Arbitrary-Command> : Should be a **valid** command we want to have the information appended to**. - The metainformation value is taken from what was last inputted with *create***. If no *create* command was inputted, it just uses an empty String. - This command **appends** the information from the last *create* command to the actual metainformation of the command. If there was no metainformation for the given command, it is just created like a *set* was called instead of *append*. -This command has integrity check so that the inputted command is actually a valid Quagga command, or at least part of it.
- **# metainfo get** <Arbitrary-Command> <Arbitrary-Command> : Should be a **valid** command we want to the metainformation from**. -** It returns onscreen what the metainformation is for the given command. - If there is no metainformation stored for the given command, a warning message as in "*No metainformation*" is shown on the CLI. -This command has integrity check so that the inputted command is actually a valid Quagga command, or at least

part of it.

- **# metainfo clear** <Arbitrary-Command> <Arbitrary-Command> : Should be a **valid** command we want to remove the metainformation from**. -** Returns warning messages (on the CLI) of success whether if there was or there was not any metainformation to be removed for the given command. - This can't be undone. -This command has integrity check so that the inputted command is actually a valid Quagga command, or at least part of it.

Helps-related functions

The following part describes every single CLI function that was included to manage the commands that handle the helps' descriptions.

Help commands

```
Router# help
append create get restore set
```

We will now proceed to give detailed information of every command for the *help* branch of commands.

- **# help create** <Arbitrary-String> <String> : Should be the data we want to store or append as a command's help. - The *create* command manages and reserves the space in memory to store the String help, its usage is completely the same as the *create* from the metainformation branch. - By itself is harmless, can handle any size of String of data. - **Only the last created string is maintained in memory for further operations**; imagine that you have a global variable that you can use anytime, and only maintains the content you last entered by the means of the *create* command.
- **# help set** <Arbitrary-Command> <Arbitrary-Command> : Should be a **valid** command we want to have its help changed. **- The help's value is taken from what was last inputted with *create***. If no *create* command was inputted, it just uses an empty String. -This command overrides whatever help value was already embed on the command, but **CAN** be restored to its original help value via *restore* CLI function. -This command has integrity check so that the inputted command is actually a valid Quagga command, or at least part of it.
- **# help append** <Arbitrary-Command> <Arbitrary-Command> : Should be a **valid** command we want to have our help string appended to**. - The help value to be appended is taken from what was last inputted with *create***. If no *create* command was inputted, it just uses an empty String. - This command **appends** the information from the last *create* command to the actual help's value of the command. Since there is always a help value from the very start of the daemon, there should be no void-case problem. -This command has integrity check so that the inputted command is actually a valid Quagga command, or at least part of it.
- **# help get** <Arbitrary-Command> <Arbitrary-Command> : Should be a **valid** command we want the help's description from. **-** It returns onscreen what the help's description is for the given command. - It just works as the '?' helper function on the CLI, but for just one

command. -This command has integrity check so that the inputted command is actually a valid Quagga command, or at least part of it.

- **# help restore** <Arbitrary-Command> <Arbitrary-Command> : Should be a **valid** command we want to have its help restored to its default state. **-** Returns the help original String to its description. - This can't be undone. -This command has integrity check so that the inputted command is actually a valid Quagga command, or at least part of it.

Persistence-related functions

Save metadata commands

```
Router# save metadata
Router# save metadata file_name
```

The following part describes every single CLI function that was included to manage the commands that handle the persistence on files.

- **# save metadata** (Optional: <file_name>) Optional: <file_name> : should be the name of the file in which the user would like to save all modified helps and inputted metainformation. - This method has its own parser and stores into a file the necessary structures for Quagga to load later again all values. - If no file_name is specified, it saves into a default file name, which is usually "Quagga_etc_dir / datahelp.dat" - There is **NO** execution of this command upon exiting, closing or restarting the daemon. If the data was not **explicitly** saved by the user, it will become a loss of data.

load metadata commands

```
Router# load metadata
 Router# load metadata file_name
```

- **# load metadata** (Optional: <file_name>) Optional: <file_name> : should be the name of a file in which the user would like to load helps and metainformation from. - Uses the same kind of parser that *save* used. - If no file_name is specified, it tries to load the default file which is usually "Quagga_etc_dir / datahelp.dat" - Differently from *save*, *load* **is positively invoked** during the boot phase of the Quagga daemon; in other words, it tries to load the default file upon starting the daemon.

12 Appendix B: Interface Specification

12.1Management Controller Module

12.1.1 Workflow Initialization

MCAddWorkflow: process and analyze a trigger in order to execute a workflow.

Inputs: trigger, log

Field	***Description***
trigger	Object containing the following information: • String authTicket: Authentication Ticket corresponding to the user session that was initiated in the GUI • String triggerTicket: Unique identifier of requested service with Timestamp • String triggerParams: JSON sentence containing all info needed by workflow processor to execute the workflow.
log	Logging information.

Return: status, log

Field	***Description***
Status	It can be 0 (request handled successfully), 1 (request handled unsuccessfully) or -1 (error).
log	Logging information.

12.1.2 Workflow Status Update

MCSetWorkflowStatus: notificates Management Controller (MC) about a new status of a workflow in execution.

Input: triggerTicket, statusCode, statusMessage, log

Field	***Description***
triggerTicket	Composed by triggerID and a timestamp.
statusCode	Status code identifier.
statusMessage	Description of the status code.

log	Logging information.

Return: 0 (ACK)

12.1.3 Management Controller Administration

- **SetDefaultCorrelationWindow:** time window to wait to check for similar workflows to arrive

Field	***Description***
[Integer Array]	new correlation window per queue

- **QueueLength:** Maximum number of concurrent jobs in the system.

Field	***Description***
Integer	New queue length

- **Preemption:** Defines the queues where it is possible to preempt workflow execution.

Field	***Description***
[Boolean Array]	List of pre-emptible queues.

12.1.4 Notification Module Administration

- **SetTriggerEndPoint:** register a new trigger end-point receiving the set of messages

Field	***Description***
Integer	Minimum Log Level to send
Protocol	Protocol used for the communication, currently only REST is supported

12.2Workflow Triggering

12.2.1 Workflow Trigger Listener Interface

Web service interface to receive information from Trigger Module

Input

Field	***Description***
TriggerTicket	Unique identifier of requested service with Timestamp

AuthTicket	Authentication Ticket corresponding to the user session that was initiated in the GUI. The Workflow trigger listener interface uses this ticket for authorization purpose.
Params	JSON String containing a map of <Key, Value> pairs defining additional trigger or service related parameters/information.

Output

Field	***Description***
Received	Integer parameter show the information was received successfully or not(0"successfully",-1"error",1"unsuccessfully")

12.3 Ontology Mapper Module

12.3.1 IP-NMS Configuration Request

This interface provides support for IP configuration management. It handles the adaptation of generic routing operations into vendor-specific configuration statements according to underlying protocols (i.e. CLI or NETCONF).

- **Input**: jsonInput - JSON String containing the following parameters:

Field	***Description***
routerVendor	Name of the vendor (e.g. Juniper, Cisco, Quagga, etc.)
routerModel	Router Model
routerIOsVersion	Version of the routers Operating System (OS)
action	Name of the requested service (e.g. *ConfigureIPAddress* - for interface configuration, *ConfigureStaticRoute* - for static route configuration, etc.)
confMode	Configuration Mode (e.g. CLI, NETCONF, etc.)
optParams	JSON string defining action-related configuration parameter values.

- **Output**: jsonOutput, JSON String containing the following parameter:

Field	***Description***
sentence	configuration sentence for the specified action and vendor

12.3.2 Ontology Mapper Administration

Interfaces to perform administrative configuration of the Ontology Mapper.

- **InitiateOfflineProcess**: Initiate or re-start offline process execution.

Field	***Description***
AuthTicket	Identifies the authentication (session)
RouterVendor	Name of the vendor (e.g. Juniper, Cisco, Quagga, etc.)
RouterModel	Router Model
OSVersion	Version of the routers Operating System (OS)
CLIHelpSetFile	XML file with command help set.

Return: 0 (offline execution successfull), 1 (offline execution unsuccessfull)

- **AddOntologyFile**: Add OWL File to Ontology Repository.

Field	***Description***
AuthTicket	Identifies the authentication (session)
OWL_Ontology	New Ontology

Return: 0 (ontology successfully added to repository), 1 (unsuccessfull addition)

- **DeleteOntologyFile**: Delete OWL File from Ontology Repository.

Field	***Description***
AuthTicket	Identifies the authentication (session)
OWLFileName	Ontology OWL file name to be deleted from the Ontology Repository.

Return: 0 (ontology successfully deleted from repository), 1 (fail to delete ontology)

- **UpdateOntologyFile**: Update OWL File from Ontology Repository.

Field	***Description***
AuthTicket	Identifies the authentication (session)
OWL_Ontology	New Ontology

Return: 0 (ontology successfully updated in repository), 1 (fail to update this ontology)

- **SaveOntologyFile**: Save OWL File to Ontology Repository.

Field	***Description***
AuthTicket	Identifies the authentication (session)
OWL_Ontology	OWL File

Return: 0 (ontology successfully saved in repository), 1 (fail to save this ontology)

- **EditOntology**: Edit existing ontology.

• ***Field***	***Description***
AuthTicket	Identifies the authentication (session)
OWLFileName	Ontology OWL file name to be deleted from the Ontology Repository.

12.4 Workflow Processor Module

12.4.1 WorkflowDatabaseService Interface

GetAssociatedWorkflow
INPUT

Field	***Description***
TriggerTicket	String: Unique TriggerID concatenated with timestamp
Log	String: Additional logging information

OUTPUT

Field	***Description***
WorkflowID	String: The corresponding ID to the triggered
Priority	Int: Current Workflow Priority
Log	String: Additional logging information

12.4.2 WorkflowDatabaseAdminstration Interface

Interfaces to add/delete/update workflow definitions within the ONE adapter

addWorkflow
INPUT

Field	***Description***
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
InterfaceSpecification	String: Interface specification in a standard format (e.g. WSDL)
WorkflowLocation	String: End-point location to identify where the workflow is executed
WorkflowDescription	Stirng: Text describing the role (operation) of the workflow
Log	String: Additional logging information

OUTPUT

Field	***Description***
Status	Integer: Defines the status of the insertion operation
WorkflowID	String: WorkflowID assigned to the inserted workflow (if successful)

Log	String: Additional logging information

updateWorkflow

INPUT

Field	*Description*
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
WorkflowID	String: WorkflowID of the workflow specification to be modified
InterfaceSpecification	String: Interface specification in a standard format (e.g. WSDL)
WorkflowLocation	String: End-point location to identify where the workflow is executed
WorkflowDescription	Stirng: Text describing the role (operation) of the workflow
Log	String: Additional logging information

OUTPUT

Field	*Description*
Status	Integer: Defines the status of the update operation
Log	String: Additional logging information

deleteWorkflow

INPUT

Field	*Description*
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
WorkflowID	String: WorkflowID of the workflow specification to be deleted
Log	String: Additional logging information

OUTPUT

Field	*Description*
Status	Integer: Defines the status of the delete operation
Log	String: Additional logging information

Interfaces to add/delete workflow-trigger associations and the associated priority

addWorkflowTriggerAssociation

INPUT

Field	*Description*
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
TriggerID	String: TriggerID for an incoming event that should be

	associated with a workflow
WorkflowID	String: WorkflowID of the associated workflow
Priority	Integer: Execution priority for scheduling in the management controller
Log	String: Additional logging information

OUTPUT

Field	***Description***
Status	Integer: Defines the status of the add operation
Log	String: Additional logging information

Note that the addWorkflowTriggerAssociation interface checks if a similar association (workflow ID and TriggerID) already exists, an in that scenario, updates the priority level based on the new request. As a result, an exclusive update interface is not included in the module.

deleteWorkflowTriggerAssociation
INPUT

Field	***Description***
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
TriggerID	String: TriggerID for an incoming event that should be associated with a workflow
WorkflowID	String: WorkflowID of the associated workflow
Log	String: Additional logging information

OUTPUT

Field	***Description***
Status	Integer: Defines the status of the delete operation
Log	String: Additional logging information

12.4.3 Workflow Initiation Interface

startWorkflowExecution
INPUT

Field	***Description***
TriggerTicket	String: TriggerTicket associated with the incoming trigger to start a workflow
AuthTicket	String: Authentication ticket to verify the user rights for workflow execution
parameters	String(JSON): Additional parameters required for workflow execution defined as a JSON string
Log	String: Additional logging information

OUTPUT

Field	***Description***
Status	Integer: Defines the status of the delete operation
Log	String: Additional logging information

12.4.4 Workflow Status Service Interface

getWorkflowStatus
INPUT

Field	***Description***
triggerTicket	String: Unique TriggerID concatenated with timestamp

OUTPUT

Field	***Description***
StatusCode	Integer: Integer parameter defining the class of the last status message received from the workflow
StatusMessage	String: Human readable status of the workflow execution
Log	String: Additional logging information

terminateWorkflow
INPUT

Field	***Description***
triggerTicket	String: Unique TriggerID concatenated with timestamp
Log	String: Additional logging information

OUTPUT

Field	***Description***
TerminationStatus	Int: 0 (success), 1(failed), -1(error)
Log	String: Additional logging information

rollbackWorkflow
INPUT

Field	***Description***
triggerTicket	String: Unique TriggerID concatenated with timestamp
Log	String: Additional logging information

OUTPUT

Field	***Description***
RollbackStatus	Int: 0(success), 1(failed), -1(error)
Log	String: Additional logging information

setWorkflowStatus

INPUT

Field	Description
triggerTicket	String: Unique TriggerID concatenated with timestamp
StatusCode	Integer: Integer parameter defining the class of the last status message received from the workflow
StatusMessage	String: Human readable status of the workflow execution
Log	String: Additional logging information

OUTPUT

Field	Description
Ok	Boolean: Reply if operation was successful
Log	String: Additional logging information

checkTermination

INPUT

Field	Description
triggerTicket	String: Unique TriggerID concatenated with timestamp

OUTPUT

Field	Description
terminationStatus	Boolean: Reply if termination was successful
Log	String: Additional logging information

checkRollback

INPUT

Field	Description
triggerTicket	String: Unique TriggerID concatenated with timestamp

OUTPUT

Field	Description
rollbackStatus	Boolean: Reply if rollback was successful
Log	String: Additional logging information

12.5Trigger Module

12.5.1 GUIEventListener Interface

ReceiveGUIEvent
INPUT

Field	***Description***
EventID	String: Unique identifier for an event originating from the GUI
AuthTicket	String: Authentication Ticket corresponding to the user session that was initiated in the GUI. The GUI Event Listener uses this ticket to confirm with the AAA if the user session is valid.
Params	String: JSON String containing a map of <Key, Value> pairs defining additional event related parameters.
Log	String: Additional logging information

OUTPUT

Field	***Description***
GUIEventReceived	Boolean: Reply if event was received successfully

12.5.2 NMSEventListener Interface

ReceiveNMSEvent
INPUT

Field	***Description***
EventID	String: Unique identifier for an event originating from the NMS
Params	String: JSON String containing a map of <Key, Value> pairs defining additional event related parameters.
Log	String
conditionRaiseTime	Long: timestamp of when the event was generated
alarmSeverity	String: Describes how important this event is
isServiceAffecting	Boolean: Describes if a service is affected by the event
alarmClass	String: Type of alarm issued in this event

OUTPUT

Field	***Description***
NMSEventReceived	Boolean: Reply if event was received successfully
Log	String: Additional logging information

12.5.3 TriggerModule Interface

ReceiveEvent
INPUT

Field	***Description***
EventID	String: Unique identifier for an event
Origin	String: Source of the event
Params	String: JSON String containing a map of <Key, Value> pairs defining additional event related parameters.
AuthTicket	String: Authentication Ticket as issued by the AAA indicating a unique user or machine authentication session
Log	String: Additional logging information

OUTPUT

Field	***Description***
ReceiveEventOk	Boolean: Reply if event was received successfully

12.5.4 TriggerModuleAdmin Interface

12.5.4.1 Event Administration

getAllEvents
INPUT

Field	***Description***
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
Log	String: Additional logging information

OUTPUT

Field	***Description***
Events	Event Array: Retrieves all events
Log	String: Additional logging information

getEvent
INPUT

Field	***Description***
eventID	String: Unique identifier for an event
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
Log	String: Additional logging information

OUTPUT

Field	***Description***
Event	Event: Corresponding Event

Log	String: Additional logging information

addEvent
INPUT

Field	***Description***
Event	Event: New Event definition
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
Log	String: Additional logging information

OUTPUT

Field	***Description***
Status	String: Reply if event was added successfully
Log	String: Additional logging information

deleteEvent
INPUT

Field	***Description***
EventID	String: Unique identifier for an event
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
Log	String: Additional logging information

OUTPUT

Field	***Description***
Status	String: Reply if event was deleted successfully
Log	String: Additional logging information

12.5.4.2 Trigger Administration

getAllTriggers
INPUT

Field	***Description***
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
Log	String: Additional logging information

OUTPUT

Field	***Description***
Triggers	Trigger Array: Retrieves all triggers
Log	String: Additional logging information

getTrigger
INPUT

Field	***Description***

TriggerID	String: Unique identifier for a trigger
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
Log	String: Additional logging information

OUTPUT

Field	***Description***
Trigger	Trigger: Corresponding Trigger
Log	String: Additional logging information

addTrigger
INPUT

Field	***Description***
Trigger	Trigger: New Trigger definition
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
Log	String: Additional logging information

OUTPUT

Field	***Description***
Status	String: Reply if trigger was added successfully
Log	String: Additional logging information

deleteTrigger
INPUT

Field	***Description***
TriggerID	String
AuthTicket	String: Authentication ticket to verify if the user has rights to perform the requested administration operation
Log	String: Additional logging information

OUTPUT

Field	***Description***
Status	String: Reply if trigger was deleted successfully
Log	String: Additional logging information

12.5.4.3 Policy Administration

getPolicy
INPUT

Field	***Description***
AuthTicket	String: Authentication ticket to verify if the user has rights to read the policy for mapping events to triggers
Log	String: Additional logging information

OUTPUT

Field	***Description***
policyString	String: Current Policy stored in the system
Log	String: Additional logging information

setPolicy

INPUT

Field	***Description***
policyString	String: Policy to be stored in the system
AuthTicket	String: Authentication ticket to verify if the user has rights to update the policy for mapping events to triggers
Log	String: Additional logging information

OUTPUT

Field	***Description***
Status	String: Reply if the policy was updated successfully
Log	String: Additional logging information

12.6IP-NMS Control Module

configureIPStaticRoute: This service configures an IP static route in the desired router.

- http://ipnms_address:port/Services/configureIPStaticRoute

Field	***Description***
RouterInfo objectiveRouter	The router to be configured information • String routerId • IPAddress managementAddress
IPStaticRoute routeParameters	The IP static route parameters to be configured • IPAddress destinationAddress • int ipv4Prefix • IPAddress gateway
String operation	The operation related to the static route configuration. Allowed values: • add • delete • delete ALL

Output:

ResponseResult rr	String result • «Success» • «Failure» String explicativeText • Variable log text: ○ «Failure when configuring router xxx» ○ «Failure when requesting Ontology Mapper translation» ○ «Router xxx not responding» ○ «Router xxx correctly configured»

configureIPLink: This service configures an IP link between two desired interfaces.

- http://ipnms_address:port/Services/configureIPLink

Field	***Description***
IPInterfaceInfo sourceIPInterface	The IP Interface needed information: • RouterInfo ownerRouter ○ String routerId ○ IPAddress managementAddress

	• String interfaceID • int logicalId
IPInterfaceInfo destinationIPInterface	The IP Interface needed information: • RouterInfo ownerRouter ○ String routerId ○ IPAddress managementAddress • String interfaceID • int logicalId
IPLinkAddressing addressing	The link addressing: • IPAddress sourceAddress • IPAddress destinationAddress • int ipPrefix
String operation	The operation related to the IP Link configuration. Allowed values: • add • delete

Output:

ResponseResult rr	String result • «Success» • «Failure» String explicativeText • Variable log text: ○ «Failure when configuring router xxx» ○ «Failure when requesting Ontology Mapper translation» ○ «Router xxx not responding» ○ «Router xxx correctly configured»

configureUNIRequest: This service configures an IP link between two desired interfaces.

- http://ipnms_address:port/Services/configureUNIRequest

Field	***Description***
RouterInfo objectiveRouter	The router to be configured information • String routerId • IPAddress managementAddress
LabelSwitchedPath lspParameters	The LSP parameters needed to configure the UNI request properly: • String lspname • IPAddress sourceAddress • IPAddress destinationAddress • LspAttributes attributes ○ String signalBandwidth ○ String switchingType ○ String gpid

	○ String encodingType • Path path ○ IPAddress[] pathAddresses ○ int pathSize
String operation	The operation related to the UNI request configuration. Allowed values: • add • delete

Output:

ResponseResult rr	String result • «Success» • «Failure» String explicativeText • Variable log text: ○ «Failure when configuring router xxx» ○ «Failure when requesting Ontology Mapper translation» ○ «Router xxx not responding» ○ «Router xxx correctly configured»

configureOffloadingThreshold: This service configures the offloading threshold for a desired link.

- http://ipnms_address:port/Services/configureOffloadingThreshold

Field	***Description***
String target	The target links: • All • Specified If target is All, no interface info is needed because it is applied to each link.
IPInterfaceInfo sourceIPInterface	The IP Interface needed information: • RouterInfo ownerRouter ○ String routerId ○ IPAddress managementAddress • String interfaceID • int logicalId
IPInterfaceInfo destinationIPInterface	The IP Interface needed information: • RouterInfo ownerRouter ○ String routerId ○ IPAddress managementAddress • String interfaceID

	◦ int logicalId
double threshold	The threshold defined in bps for the offloading

Output:

ResponseResult rr	String result • «Success» • «Failure» String explicativeText • Variable log text: ◦ «Failure when configuring router xxx» ◦ «Failure when requesting Ontology Mapper translation» ◦ «Router xxx not responding» ◦ «Router xxx correctly configured»

configureRollbackOffloadingThreshold: This service configures the rollback offloading threshold for a desired link.

- http://ipnms_address:port/Services/configureRollbackOffloadingThreshold

Field	***Description***
String target	The target links: • All • Specified If target is All, no interface info is needed because it is applied to each link.
IPInterfaceInfo sourceIPInterface	The IP Interface needed information: • RouterInfo ownerRouter ◦ String routerId ◦ IPAddress managementAddress • String interfaceID • int logicalId
IPInterfaceInfo destinationIPInterface	The IP Interface needed information: • RouterInfo ownerRouter ◦ String routerId ◦ IPAddress managementAddress • String interfaceID ◦ int logicalId
double threshold	The threshold defined in bps for the rollback operation in offloading

Output:

ResponseResult rr	String result • «Success»

	• «Failure» String explicativeText • Variable log text: ○ «Failure when configuring router xxx» ○ «Failure when requesting Ontology Mapper translation» ○ «Router xxx not responding» ○ «Router xxx correctly configured»

getBandwidthInformation: This service allows other modules to request for bandwidth information. The response is a double with the bps measured by the IPNMS control module.

- http://ipnms_address:port/Services/getBandwidthInformation

Field	***Description***
IPInterfaceInfo sourceIPInterface	The IP Interface needed information: • RouterInfo ownerRouter ○ String routerId ○ IPAddress managementAddress • String interfaceID • int logicalId
IPInterfaceInfo sourceIPInterface	The IP Interface needed information: • RouterInfo ownerRouter ○ String routerId ○ IPAddress managementAddress • String interfaceID ○ int logicalId

configureAuthInfo: This service configures router authentication info.

- http://ipnms_address:port/Services/ configureAuthInfo.

RouterInfo objectiveRouter	The router to be configured information • AuthenticationInfo: ○ String username ○ String password • String routerId • IPAddress managementAddress

Output:

ResponseResult rr	String result • «Success» • «Failure»

	String explicativeText • Variable log text: ○ «Failure when configuring router xxx» ○ «Failure when requesting Ontology Mapper translation» ○ «Router xxx not responding» ○ «Router xxx correctly configured»

Responses

All operations will be answered with the following response commands.

ResponseResult rr	String result • «Success» • «Failure» String explicativeText • Variable log text: ○ «Failure when configuring router xxx» ○ «Failure when requesting Ontology Mapper translation» ○ «Router xxx not responding» ○ «Router xxx correctly configured»

12.7 T-NMS Module

12.7.1 SNMP Binding Connector

Main purpose of this module is to collect all necessary notifications from SNMP channel, map them to the ONE internal topology, and forward them to the Trigger Module.

12.7.2 Notification Server Administration

Adding address for SNMP notification handling

Field	***Description***
Addresses	List of network addresses to connect that provide SNMP service
Current Conditions Flag	Flag that decides whether server should fetch current conditions a push them into topology module

Removing address from notification handling

Field	***Description***
Addresses	List of network addresses to stop listening

12.8OpenFlow Control Module

In OpenFlow service is mapped to a single flow between two ports. In Transport Networks endpoints will be described as ports. After receiving request controller will try to create a service by OpenFlow Agent and if succeed will add flow to a table. Automatic update of topology should be performed shortly.

12.8.1 OpenFlow Connector Administration

Due to Floodlight Controller does not provide publish/subscribe interface module must check periodically whether configuration has changed or not since last query, and if applicable push data into the Topology Module.

Interface for adding address for OpenFlow connector

Field	***Description***
Addresses	List of network addresses to connect
Force flag	Boolean value that determines whether current topology data should be retrieved even it seems to be up to date

Flow and Port retrieval could be also done after certain conditions. For example if service has been created topology module should get up to date flow list.

For removing address

Field	***Description***
Addresses	List of network addresses to disconnect

12.8.2 Creating a Service

In OpenFlow service is mapped to a single flow between two ports. In Transport Networks endpoints will be described as ports. After receiving request controller will try to create a service by OpenFlow Agent and if succeed will add flow to a table. Automatic update of topology should be performed shortly after.

INPUT

Field	***Description***
Src Definitions	Switch ID and integer that identifies source port
Dst Definitions	Switch ID and integer that identifies destination port
Controller Address	IP address that identifies controller
FlowName	String that will allow to find service in flow table later

OUTPUT

Field	***Description***
Boolean	Flag set to true if service creation succeed
Message	Allows to describe cause of unsuccessful service creation

12.8.3 Service Deletion

Field	***Description***
Controller Address	IP address that identifies controller
FlowName	String that will allow to find service in flow table

12.9Topology Module

UpdateNode: This function inserts the node in the graph. If the node is not in the graph it is inserted, and if the node is in the graph, it is replaced for the new one.

Input:

- http://URL/TopologyModule/UpdateNode
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
NodeName	Node name, it is unique
Address	Node address
IsPhysical (optional field)	It is a Boolean field specifying if the node is physical or not
Domain	Node domain
LayerNode	Layer where the node is
ParentRouter (optional field)	Parent Router
xLocation (optional field)	Coordinate x where the node is placed
yLocation (optional field)	Coordinate y where the node is placed

Output:

- String:
  - True if succeeded or false otherwise

UpdateLink: This function inserts the link between the source and destination nodes, connected to the interfaces. The nodes and interfaces must exist.

Input:

- http://URL/TopologyModule/UpdateLink
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
linkID	Identifier Link, it is unique
IsDirectional (optional field)	It is a Boolean field specifying if the link it is directional or not
srcNode	Node name of the source node
srcIntf	Interface of the source node to which the link will be connected
dstNode	Node name of the destination node
dstIntf	Interface of the destination node to which the link will be connected
LayerLink	Layer to which the link owns. It could be 'transport', 'IP' or 'interlayer'
TypeLink (optional field)	Field indicating the type link: *intradomain, interdomain* or *interlayer*
teMetric (optional field)	Link metric

Output:

- String:
  - True if succeeded or false otherwise

UpdateIntf: This function inserts the interface in the node. The node must exist. If the interface exists in the node it is replaced.

Input:

- http://URL/TopologyModule/UpdateIntf
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
NodeName	Node name, it is unique
IntfName	Interface name
Address	Interface address
Layering	Layer in which the interface is. It could be 'transport' or 'IP'
IsPhysical (optional field)	Boolean field indicating if the interfaces is physical or not
intfUp (optional field)	Boolean field indication if the interface is up or down

Output:

- String:
  - True if succeeded or false otherwise

DeleteNode: deletes the node from the graph.

Input:

- http://URL/TopologyModule/DeleteNode
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
NodeName	Node name, it is unique

Output:

- String:
  - True if succeeded or false otherwise

DeleteLink: deletes the link from the graph.

- http://URL/TopologyModule/DeleteLink
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
linked	Identifier Link, it is unique

Output:

- String:
  - True if succeeded or false otherwise

DeleteIntf: deletes the link from the graph.

- http://URL/TopologyModule/DeleteIntf
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
NodeName	Node name, it is unique
IntfName	Interface name

Output:

- String:
  - True if succeeded or false otherwise

GetFullTopology: Retrieves all the topology information of a given layer or of the three layers.

- http://URL/TopologyModule/GetFullTopology
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
Layer (optional field)	Layer of the topology returned
DomainID	Domain of the topology returned

Output:

- A JSON String with the topology structure.

- Example:

{"nodeID":"ADVA_NODE_4","address":["172.16.1.40"],"isPhysical":true,"intfList":[{"name":"ADVA_NODE_4_Shelf1-13","address":["41.41.41.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_4_Shelf1-10","address":["40.40.40.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_4_Unnumbered1","address":["1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_4_Unnumbered2","address":["2"],"layering":["transport"],"isPhysical":true,"intfUp":true}],"domain":1,"layer":"transport"}{"nodeID":"ADVA_NODE_2","address":["172.16.1.36"],"isPhysical":true,"intfList":[{"name":"ADVA_NODE_2_Unnumbered1","address":["1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_2_Unnumbered2","address":["2"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_2_Unnumbered3","address":["3"],"layering":["transport"],"isPhysical":true,"intfUp":true}],"domain":1,"layer":"transport"}{"nodeID":"ADVA_NODE_3","address":["172.16.1.38"],"isPhysical":true,"intfList":[{"name":"ADVA_NODE_3_Shelf1-

7","address":["31.31.31.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_3_Shelf1-14","address":["30.30.30.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_3_Unnumbered1","address":["1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_3_Unnumbered2","address":["2"],"layering":["transport"],"isPhysical":true,"intfUp":true}],"domain":1,"layer":"transport"}{"nodeID":"ADVA_NODE_1","address":["172.16.1.34"],"isPhysical":true,"intfList":[{"name":"ADVA_NODE_1_Shelf1-11","address":["21.21.21.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_1_Shelf1-9","address":["20.20.20.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_1_Unnumbered1","address":["1"],"layering":["transport"],"isPhysical":true,"intfUp":true}],"domain":1,"layer":"transport"}{"linkID":"ADVA_NODE_4_ADVA_NODE_3","isDirectional":true,"source":{"node":"ADVA_NODE_4","intf":"ADVA_NODE_4_Unnumbered1"},"dest":{"node":"ADVA_NODE_3","intf":"ADVA_NODE_3_Unnumbered1"},"type":"intradomain","teMetric":0.0}{"linkID":"ADVA_NODE_4_ADVA_NODE_2","isDirectional":true,"source":{"node":"ADVA_NODE_4","intf":"ADVA_NODE_4_Unnumbered2"},"dest":{"node":"ADVA_NODE_2","intf":"ADVA_NODE_2_Unnumbered1"},"type":"intradomain","teMetric":0.0}{"linkID":"ADVA_NODE_3_ADVA_NODE_2","isDirectional":true,"source":{"node":"ADVA_NODE_3","intf":"ADVA_NODE_3_Unnumbered2"},"dest":{"node":"ADVA_NODE_2","intf":"ADVA_NODE_2_Unnumbered2"},"type":"intradomain","teMetric":0.0}{"linkID":"ADVA_NODE_2_ADVA_NODE_1","isDirectional":true,"source":{"node":"ADVA_NODE_2","intf":"ADVA_NODE_2_Unnumbered3"},"dest":{"node":"ADVA_NODE_1","intf":"ADVA_NODE_1_Unnumbered1"},"type":"intradomain","teMetric":0.0}

GetOppositeNode: Retrieves the node in the other layer to which the interface is connected.

Input:

- http://URL/TopologyModule/GetOppositeNode
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
intfName	Interface name
Address	Interface address
Layer	Layer in which the interface is. It could be 'transport' or 'IP'
IsPhysical	Boolean field indicating if the interfaces is physical or not
ParentIntfName	Parent interface name

Output:

- JSON String with the node and the interface names.
- Example: {"node":"ADVA_NODE_4","intf":"ADVA_NODE_4_Shelf1-13"}

GetOppositeInterface: Retrieves the interface in the other layer to which the interface is connected.

Input:

- http://URL/TopologyModule/GetOppositeInterface
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
intfName	Interface name
Address	Interface address
Layer	Layer in which the interface is. It could be 'transport' or 'IP'
IsPhysical	Boolean field indicating if the interfaces is physical or not
ParentIntfName (optional)	Parent interface name

Output:

- JSON String with the name of the opposite interface.

- Example: "ADVA_NODE_4_Shelf1-13"

GetNeighboursNodesOf: Retrieves the neighbour nodes of a given node in a layer.

- http://URL/TopologyModule/GetNeighboursNodesOf
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	***Description***
— NodeName	— Node name

Output:

- A JSON String with the list of the node neighbours structure. The fields are:
  - NodeID: node name.
  - Address: node ip address.
  - IsPhysical: string (true or false).
  - IntfList: list of interface structure of the node. Containing the following fields:
    - Name: interface name.
    - Address: interface ip address
    - Layering: 'Transport' or 'IP'
    - isPhysical: true or false.
    - intfUp: Indicates if the interface is up or down. Value: true or false.
  - domain: node domain
  - location (optional): position where the node is located.
  - parentRouter (optional)
  - layer: 'transport' or 'IP'.
- Example:

{"nodeID":"ADVA_NODE_4","address":["172.16.1.40"],"isPhysical":true,"intfList":[{"name":"ADVA_NODE_4_Shelf1-13","address":["41.41.41.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_4_Shelf1-10","address":["40.40.40.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_4_Unnumbered1","address":["1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_4_Unnumbered2","address":["2"],"layering":["transport"],"isPhysical":true,"intfUp":true}],"domain":1,"layer":"transport"}{"nodeID":"ADVA_NODE_3","address":["172.16.1.38"],"isPhysical":true,"intfList":[{"name":"ADVA_NODE_3_Shelf1-7","address":["31.31.31.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_3_Shelf1-14","address":["30.30.30.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_3_Unnumbered1","address":["1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_3_Unnumbered2","address":["2"],"layering":["transport"],"isPhysical":true,"intfUp":true}],"domain":1,"layer":"transport"}{"nodeID":"ADVA_NODE_1","address":["172.16.1.34"],"isPhysical":true,"intfList":[{"name":"ADVA_NODE_1_Shelf1-11","address":["21.21.21.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_1_Shelf1-9","address":["20.20.20.1"],"layering":["transport"],"isPhysical":true,"intfUp":true},{"name":"ADVA_NODE_1_Unnumbered1","address":["1"],"layering":["transport"],"isPhysical":true,"intfUp":true}],"domain":1,"layer":"transport"}

MeasurementModule: It returns the layer of the interface name received by argument.

Input:

- http://URL/TopologyModule/MeasurementModule
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	*Description*
IntfName	Interface name

Output:

- String:
    - 'Transport' or 'IP': if the interface exists.
    - Null: if the interface does not exist

GetNodeByName: Returns the node.

Input:

- http://URL/TopologyModule/GetNodeByName
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	*Description*
NodeName	Node name

Output:

- A JSON String with the node structure. The fields are:
    - NodeID: node name.
    - Address: node ip address.
    - IsPhysical: string (true or false).
    - IntfList: list of interface structure of the node. Containing the following fields:
        - Name: interface name.
        - Address: interface ip address
        - Layering: 'Transport' or 'IP'
        - isPhysical: true or false.
        - intfUp: Indicates if the interface is up or down. Value: true or false.
    - domain: node domain
    - location (optional): position where the node is located.
    - parentRouter (optional)
    - layer: 'transport' or 'IP'.

Example:

{"nodeName":"MX240-3"}

{"nodeID":"MX240-3","address":["192.168.8.3"],"isPhysical":true,"intfList":[{"name":"MX240-3_ge-2/1/8","address":["21.21.21.2"],"layering":["IP"],"isPhysical":true,"intfUp":true},{"name":"MX240-3_ge-2/1/9","address":["20.20.20.2"],"layering":["IP"],"isPhysical":true,"intfUp":true}],"domain":1,"layer":"IP"}

GetIntfByName: Returns the interface.

Input:

- http://URL/TopologyModule/GetIntfByName
- JSON String containing a Map (<Key, Value>) containing the following parameters:

Field	*Description*
IntfName	Interface name

Output:

- String:
    - 'Transport' or 'IP': if the interface exists.

  - Null: if the interface does not exist

12.10 Measurement Module Interface

12.10.1 Get PM Data Interface Current Data

INPUT

Field	***Description***
Name	Requested interface name containing all infromation
PmType	Name of requested PMData
Scope	Granularity of data (24h, 15min etc.)

OUTPUT

Field	***Description***
Name	Interface name
PmType	Name of requested PMData
Scope	Granularity of data (24h, 15min etc.)
Value	PMData value

12.10.2 Get PM Data Interface Historical Data

INPUT

Field	***Description***
Name	Requested interface name
PmType	Name of requested PMData
Scope	Granularity of data (24h, 15min etc.)
StartDate	Date for oldest data
EndDate	Date for youngest data

OUTPUT

Field	***Description***
Name	Interface name
PmType	Name of requested PMData
Scope	Granularity of data (24h, 15min etc.)
Value	PMData value
StartDate	Date for oldest data
EndDate	Date for youngest data

12.11 PCE Module

12.11.1 PathComputationElementModule Interface

reqMplsPath
INPUT

Field	***Description***
triggerTicket	String: Unique TriggerID concatenated with timestamp
srcIPAddr	String: Source Address
destIPAddr	String: Destination Address
Bw	Double: Bandwidth demand
metricList	<Int, Double> List defining the constraints on different metric types for path computation
Ofcode	Int: Objective Function code that enforces specific PCE path computation algorithm
Log	String: Additional logging information

OUTPUT

Field	***Description***
compSuccess	Boolean: Reply if path could be found
Path	Path Array: Array of possible paths from source to destination The complex Path type is structured as follows: • MetricList: <Int, Double> List defining the different metric types and the corresponding parameter value • Bw: Bandwith parameter • Ero: Explicit Route Object that represent the path
Log	String: Additional logging information

12.12 AAA Module

12.12.1 User Authentication Interface

Web service interface to receive user information from any module

Input

Field	***Description***
UserName	String that contains username.
Password	MD5 String that contains password.

Output

Field	***Description***
AuthTicket	Authentication Ticket corresponding to the user session that was initiated in the GUI. The Workflow trigger listener interface uses this ticket for authorization purpose.
Adminlevel	Show the type of user,User can edit information or just view or administrator
Network	String containing network information

12.12.2 Trigger Authentication Interface

Web service interface to receive Authentication information.

Input

Field	***Description***
AuthTicket	Authentication Ticket corresponding to the user session that was initiated in the GUI.

Output

Field	***Description***
Valid	Boolean parameter show the given authticket is valid or not.

12.12.3 Workflow Authorization Interface

Web service interface to receive information from the Workflow Triggering Module

Input

Field	***Description***
TriggerTicket	Unique identifier of requested service with Timestamp
AuthTicket	Authentication Ticket corresponding to the user session that was initiated in the GUI.

Output

Field	***Description***
Valid	Integer parameter show the given authticket and triggertricket are authorized and valid or not (0"valid",-1"error",1"invalid")

12.12.4 Accounting Interface

Web service interface to receive information from Smart Analytics Module for initialization/termination of accounting

AccountingStart : Initialize accounting

Input

Field	***Description***
TriggerTicket	Unique identifier of requested service with Timestamp
AuthTicket	Authentication Ticket corresponding to the user session that was initiated in the GUI.
Time	Starting time of accounting (Long type)

Output

Field	***Description***
AccountingTicket	String parameter contains the accounting ticket information,if the triggerticket and authticket are valid it will be create otherwise it shows not valid.

AccountingStop : Terminate accounting

Input

Field	***Description***
AccountingTicket	Unique identifier of Accounting that generates based on Trigger Ticket and Authentication Ticket
Time	Termination time of accounting (Long type)

Output

Field	***Description***
Returnvalue	Integer paramerer shows the termination handled successfully or not(0"successfully",-1"error",1"unsuccessfully")

12.12.5 Administration Interface

Input

Field	***Description***
AuthTicket	Authentication Ticket corresponding to the user session that was initiated in the GUI.
Module	Module on which the administration operation is to be performed
Operation	Associated administration operation on the module

Output

Field	***Description***
authorizationStatus	Boolean to indicate if the user is allowed to perform the requested operation.
log	Additional logging information.