



Deliverable D4.13

Integrated demonstration of a combination of use-case prototypes

Grant Agreement	257513	
Date of Annex I	08 October 2013	
Dissemination Level	Public	
Nature	Report	
Work package	WP4 – Deployment and Impacts	
Due delivery date	01 August 2013	
Actual delivery date	21 October 2013	
Lead beneficiary	NKUA	Roi Arapoglou (roiar@di.uoa.gr)

Authors	ALBLF: Pierre Peloso, Laurent Ciavaglia NKUA: George Katsikas, Roi Arapoglou, Eleni Patouni, Nancy Alonistioti UPRC: Panagiotis Demestichas, Kostas Tsagkaris, Aristi Galani, Vasilis Foteinos, Vera Stavroulaki, Giorgios Poullos, Yiouli Kritikou, Marios Logothetis, Dimitris Karvounas, Andreas Georgakopoulos, Assimina Sarli, Evangelia Tzifa, Nikolaos Koutsouris, Petros Morakos, Alexandros Antzoulatos, Aimilia Bantouna UCL: Alex Galis, Stuart Clayman, Lefteris Mamatas VTT Teemu Rautio
----------------	---

Executive summary

This deliverable describes the UniverSelf Integrated Demonstration entitled ‘Integrated demonstration of a combination of use-case prototypes’, in terms of deployment and implementation aspects of UMF core blocks functionality, respective NEMs, NEMs skin and integrated data planes. It should be noted that the Integrated Demonstration has been successfully showcased in Future Network & Mobile Summit 2013 (FUNEMS 2013 [9]).

The UniverSelf Integrated Demonstration comprises the demonstration of three scenarios so as to highlight different UMF aspects. In each scenario different and heterogeneous network domains exist where the deployment of different NEMs takes place so as to solve real life operator/network problems.

The results validated the deployment of UMF core blocks, in terms of performance and efficiency and highlight the “unified” nature of UMF. More details on the different prototype components are available in the respective deliverables for the three use case prototypes (D4.4/D4/5 [17][18], D4.8/D4.9[19][20] and D4.10/D4.11[21][22]) and deployment results ([23]).

In addition, guidelines and instructions for installing, configuring and running UMF software are given.

Elaborating upon the integrated demonstrator, the final prototype will deliver a single-story scenario capturing all the main project aspects and will be showcased within the project final review (November 2013).

Table of Content

1	Introduction	5
2	Integrated Demonstration Description	7
2.1	UMF Core	7
2.2	NEM SKIN	7
2.2.1	Functionalities	8
2.2.2	Interfacing	9
2.3	Integrated Demonstration Dataplanes	9
2.3.1	Governance	9
2.3.2	Coordination.....	12
2.3.3	Knowledge.....	13
3	Integrated Demonstration Scenarios.....	15
3.1	A Unified Framework for QoS and SLA-aware multi-domain self-management.....	15
3.1.1	Multi-domain testbed description	15
3.1.2	Heterogeneous NEM deployment.....	16
3.1.3	Scenario Storyline.....	18
3.1.4	Summary	22
3.2	Conflict-free Coordination of SON NEMs.....	23
3.2.1	Scenario Storyline.....	23
3.2.2	Results	26
3.2.3	Summary	27
3.3	UMF Management of Software Defined Networks	28
3.3.1	VLSP Overview.....	28
3.3.2	UMF management of Software Defined Networks (SDN).....	33
3.3.3	Summary	37
4	Conclusions	38
5	Annex A	39
5.1	Integrated demonstration and future steps	39
6	Annex B	40
6.1	Instructions on building and running the Latest UMF Core Blocks and NEM prototypes	40
6.1.1	Setting up the simulator	40
6.1.2	Building the sources	43
6.1.3	Running the prototype	43
6.1.4	Installation instructions of the Knowledge core service	43
7	References	46
8	Abbreviations.....	47

1 Introduction

This deliverable analyses the outcome of the deployment and implementation work within WP4, delivering the integrated demonstration of a combination of use-case prototypes. Specifically the integrated test-bed will demonstrate the interworking of the use-case prototypes described in D4.4/D4.5 [17][18], D4.8/D4.9[19][20] and D4.10/D4.11[21][22], in the context of given scenarios. In detail the integrator demonstrator will deliver the combinations of the first, second and third use-case prototype, as it was presented in the FUNEMS 2013 UniverSelf exhibition [9]). It should be noted that the described prototype has formed the basis for the final UniverSelf prototype that will be delivered and presented during the Y3 review. The evolution path towards the final prototype is focused on the full-alignment with the updated version of the NEM skin, the fine-tuning of the implemented UMF functions according to the UMF final release specification and the full integration of the developed components and software modules under a common scenario.

D4.13 elaborates on the integrated demonstrator, focusing on the deployment of UMF Core Blocks, the NEMs, supporting mechanisms and the integrated data planes as well as the demonstration of a set of scenarios; the goal is to highlight the unified nature of UMF. Specifically, section 2 provides a detailed description of the UMF core Blocks (namely Governance, Knowledge and Coordination) from the implementation point of view, focusing the deployed functionality. This analysis is complemented with the analysis of the NEM skin, in terms of functionalities and interfaces. Finally the three integrated data planes are presented in detail under the three following axes: the triggering of business level policies, the identification of conflicting behaviour between instantiated NEMs and the control of knowledge exchange between NEMs. More details with respect to performance and efficiency issues are available in D4.12 and D4.15.

Furthermore, three integrated demonstration scenarios are presented in Section 3: namely, “A Unified Framework for QoS and SLA-aware multi-domain self-management”, “Conflict-free Coordination of SON NEMs” and “UMF Management of Software Defined Networks”. Within these scenarios, totally 7 NEMs have been deployed and integrated, sharing the same UMF Core Block functionalities. In the first scenario, WLAN Infrastructure Management NEM, Wireless access Load Balancing NEM, Virtual Infrastructure Management NEM and Core Load Balancing NEM are deployed upon a heterogeneous network topology [25]. The main purpose of first scenario is to showcase the existence of a QoS based load balancing mechanism that is both enforced on the access and the core part of existing topology. The second scenario presented an LTE HetNet with Self-Organizing Network (SON) functions that are deployed within a respective NEM. In the context of this scenario, the Load Balancing NEM and BackhaulOptimization NEM are deployed [11]. The purpose of the second scenario is to present conflict-identification and resolution functions of the UMF COO block. The third scenario presents the UMF management of Software Defined Networks, showcasing the deployment of the Virtual Infrastructure NEM [26].

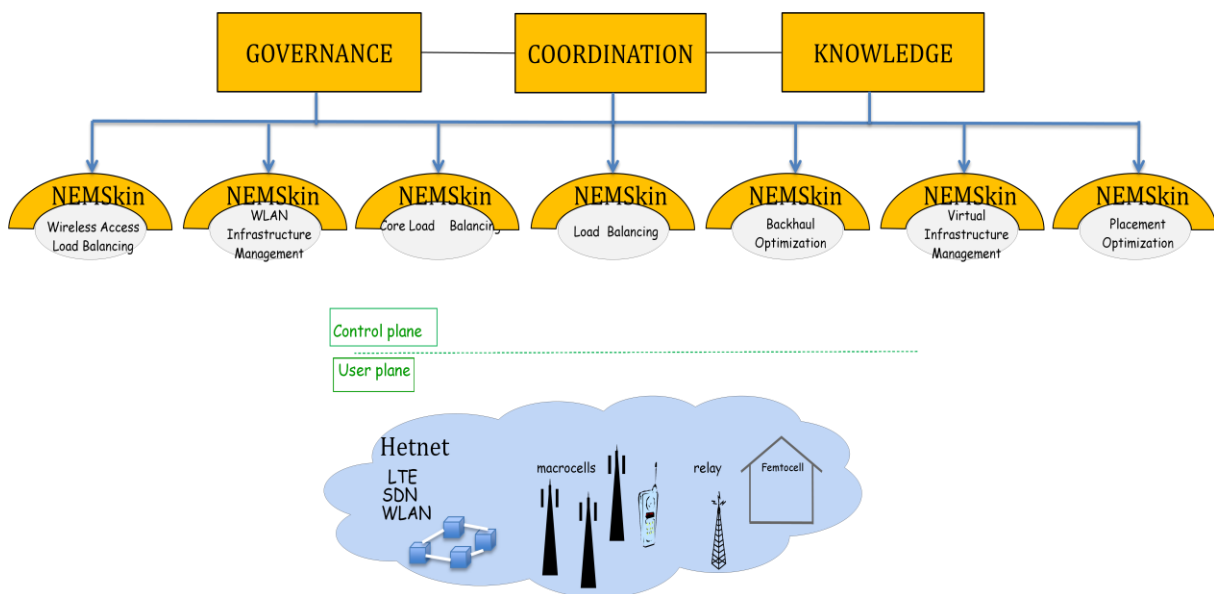


Figure 1 Integrated Demonstration

Finally, Chapter 4 concludes the deliverable by summarising the outcomes of the integrated demonstrator comprising the three use-case prototypes. It should be noted that the ANNEX part of the document provides future NEM developers with a useful guide so as to download, install and run the integrated prototype.

2 Integrated Demonstration Description

The purpose of this integrated demonstration of the use case prototypes is to investigate the real life deployment of UMF as a Network Operator tool for the efficient orchestration of a complex heterogeneous network environment. Towards this direction, the instantiation and deployment of a series of NEMs on top of an existing topology is presented, which constitute a key solution to diverse real-life problems of operators/networks.

The considered network topology comprises the following testbed and respective components:

- The wireless access network which consists of three soekris net5501 devices, one for backhauling and two for user connectivity, as well as three classes of user terminals that are connected to the access.
- The core network is comprised of a Software Defined Virtual Network that acts as one segment of a full network where video content passes through it
- Simulated components for macro- and relay stations are provided for the coordination of two SON functionalities in a LTE-Advanced heterogeneous network.

The rest of this section provides an overview of the implemented components, focusing on the UMF Core Blocks, the NEM SKIN and a detailed presentation of Integrated Demonstration Data planes.

2.1 UMF Core

UMF is a management framework that aims to solve actual network problems and address the growing management complexity of future networks, through the embodiment and empowerment of autonomic principles and techniques in both services and networks. UMF manages networks and services through the management of NEMs. A NEM encapsulates autonomic functions, and it is formally defined as a functional grouping of objective(s) + context + method(s) where “method” is a general procedure for solving a problem. A NEM is (a priori) implemented as a piece of software that can be deployed in a (part of a) network to enhance/simplify its control and management (e.g. take over some operations). An example of NEM is the use of genetic algorithm (the method) for interference coordination (the objective) in LTE networks (the context).

UMF enables trustworthy integration and interworking of NEMs within the operator's management ecosystem. The achievement of this objective requires the governance of NEM's, the orchestration/coordination of different NEMs' behaviour, as well as information/knowledge sharing among NEMs. This demand translates into the introduction of the three UMF core blocks, Governance (GOV), Knowledge (KNOW) and Coordination (COORD), depicted in Figure 2.

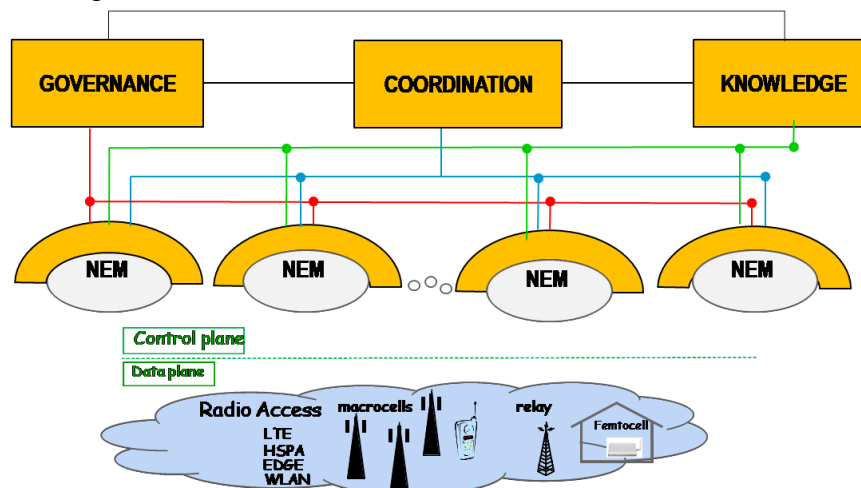


Figure 2: Schematic representation of UMF core blocks

2.2 NEM SKIN

The NEM skin is the software base for any NEM. In particular, using Object-Oriented Design (OOD) terminology, it literally is a superclass to a NEM class. It includes functionality common to all NEMs and provides the

interfacing (tools and implementation) with the UMF Core blocks. A functional overview of this software component is depicted in Figure 3 and a more detailed description is provided in the following paragraphs. Although it is a complementary component for NEM development, noteworthy is the fact that it assisted and was used by all NEMs demonstrated in the final, integrated prototype.

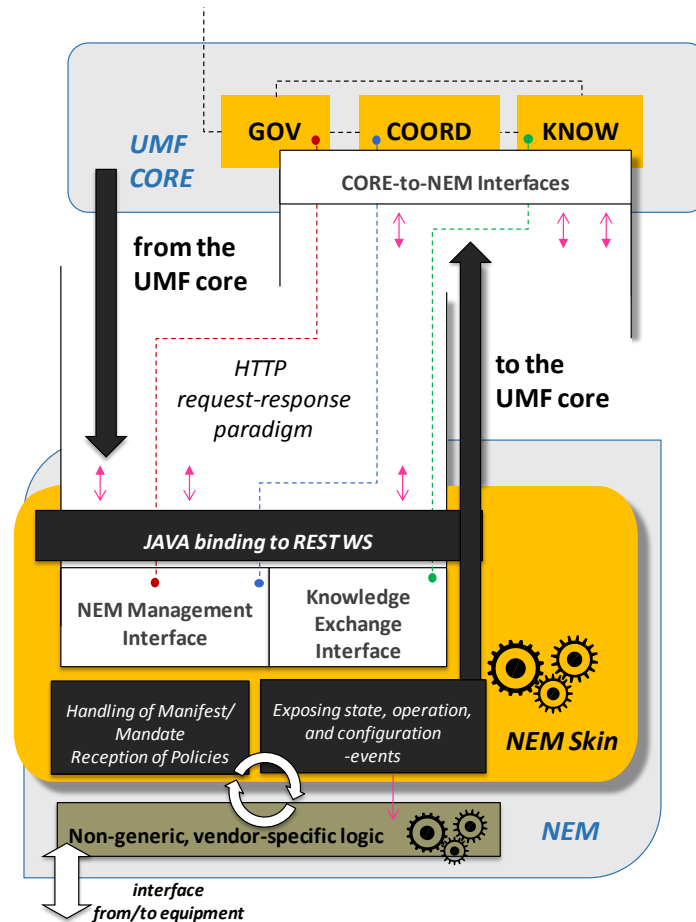


Figure 3: Functional Overview of the NEM Skin

2.2.1 Functionalities

The NEM skin is responsible for receiving the Mandate [2][24], and maintaining an up-to-date, sane, local copy. From there on, it undertakes to update the list of managed entities and invokes deploy/undeploy methods on its subclass (the NEM), where the NEM developer has implemented the vendor-specific logic. Respectively similar responsibilities apply also for the Policies/Configuration options. Additionally, for the NEM Regime policies, the NEM skin interprets them into corresponding invocations of the monitor/analyze/plan/execute methods on the NEM class. This particular task involves the usage of timers and threads to maintain the required periodicity dictated by the regime. The NEM developer is therefore exempt from the (potentially error-prone) task of analyzing the regime and driving his NEM's MAPE loop accordingly.

Furthermore, since all NEMs are statefull entities, the NEM skin ensures the appropriate state transitions and maintenance of a STATE variable (synchronization to ensure a sane STATE value in a multi-threaded, multi-process environment like UMF can be a significantly bug-fertile task). For instance, a registration is triggered after deployment, and if successful, the STATE variable will be set to REGISTERED, otherwise failure will be returned for the anticipated cases or appropriate exceptions will be thrown. As should be obvious from this example, the NEM skin implements the NEM-related-part of the UMF workflows by invoking the appropriate methods, whether these are implemented in the skin or the NEM itself (a polymorphism capability provided by most OO languages using method overriding). The added-value in this case is that the NEM developer is exempt from the effort of achieving UMF compliancy, at least in the workflow level.

Helper/convenience functionality is also included in the NEM skin for easy knowledge block interactions and knowledge exchange, abstracting the negotiation phase and the decisions to be taken during the course of flow establishments for the NEM developer.

Last but not least, the NEM skin maintains a list of event listeners (either external/other instances or internal/subclasses, whether remote or local) and generates events towards them regarding its state transitions, operation taking place (registration, deployment etc) and application of configuration options. Remarkably, one of those listeners is the UMF dashboard so as to display the current state of the NEM. From the NEM developer perspective, it allows them to easily intervene at any stage of the NEM lifecycle.

2.2.2 Interfacing

Communication between NEMs and the UMF Core takes place over the network, and in particular using the HTTP protocol. Yet, all networking details (URI resource handling, HTTP details, data serialization and deserialization, server and client management) are completely transparent to the NEM developer. HTTP mechanics are performed by the NEM skin to enable seamless integration of the NEM class with the UMF Core. For that purpose, it employs an RMI-over-HTTP framework, which in turn utilizes other fast and lightweight web server/client, and serialization libraries for Java. The end result is compile-time binding (therefore less and easier debugging time), clean and simple Java code (e.g. registering to GOV is as simple as a `getGov().register(...)` line of code, which is in fact the case), and rapid integration as soon as the Java interfaces are in place.

2.3 Integrated Demonstration Data planes

An extensive description of Integrated Demonstration Data planes as these have been presented in FUNEMS 2013 demonstration is provided in this section. For every specific UMF Core Block the corresponding data planes are described, focusing on the triggering of business level policies, the identification of conflicting behaviour between instantiated NEMs and the control of knowledge exchange between NEMs.

2.3.1 Governance

The Human to Network function provides a friendly tool for the human operator to dynamically define high-level objectives and user requirements, which will be later on translated into technology-specific terms. This function alleviates the human operator from the need to deal with technical details. High-level objectives may include definition of new services, user classes (gold, silver, bronze) and their QoS levels etc. In a subsequent step, the high level objectives are forwarded to the Policy Derivation & Management (PDM) module, as this has been already introduced in [24].

At the same time, the Human to Network function provides means for the human operator to receive information about of the functioning of the network, services and the UMF system.

In order to achieve these goals, the Human to Network Interface provides a set of interfaces (screens) to the operator; such interfaces are implemented inside the Policy Derivation and Management module. The aforementioned functionality corresponds to the defined interfaces:

- Ontology Concept Composition, which provides the operator with the ability to define his own high level ontology concept language (meta-language).
- Policy Composition, which allows the specification of operator's high level business goals: type of service, service level agreement, network technologies, processes which will run upon a considered topology, high level goals etc.
- Policy Activation, which allows the selection of appropriate high-level business objectives (already composed in Policy Composition module) and the priority levels, i.e. in case two or more policies apply at a specific network domain, the priority level defines which goal should be fulfilled first.
- NEM Conflicts, which inform the operator on the outcome of the policy translation process, in terms of NEMs that do not fulfil high-level goals.
- Available NEMs, which provides the operator with already registered NEMs' manifests.

In UniverSelf, the Human to Network function has been implemented in Java, with a Graphical User Interface to allow the interactions with the human operator. Illustrative screenshots of the Service Definition and Supervision operations are shown in Figure 4 and Figure 5 respectively.

Ontology Concept	Policy Composition	Policy Activation	NEM Conflicts	Database Management	Available NEMs
------------------	--------------------	-------------------	---------------	---------------------	----------------

Ontology Concept

Service SLA				Process		Network Domains & Technology			Goal	
Service SLA concepts				Process concepts		Network Domains & Technology concepts			Goal concepts	
TYPE SERVICE	TYPE	VALUES		CONCEPT NAME		CONCEPT NAME	TECHNOLOGY		GOAL	
Video	QoS	Critical Normal Excellent Any	<input type="button" value="Del"/>	Monitoring	<input type="button" value="Del"/>	Access	WLAN	<input type="button" value="Del"/>	Load Balancing	<input type="button" value="Del"/>
				Reconfiguration	<input type="button" value="Del"/>	Core	Fixed IP	<input type="button" value="Del"/>	Energy Efficiency	<input type="button" value="Del"/>
Any	Any	Any	<input type="button" value="Del"/>						Any	<input type="button" value="Del"/>

Add Service SLA Concept

Type Service:

Type SLA:

Choose qualities:

☐ Critical
 ☐ Minimum
☐ Normal
 ☐ Good
☐ Excellent
 ☐ Any

Add Process Concept

Type Process:

Add Network Domains & Technology Concept

Type Network Domain:

Type Technology:

Add Goal Concept

Type Goal:

Ontology Concept	Policy Composition	Policy Activation	NEM Conflicts	Database Management	Available NEMs
------------------	--------------------	-------------------	---------------	---------------------	----------------

Policy Activation

Activate/Deactivate created policies

ID	SLA	PROCESS	NETWORK DOMAINS & TECHNOLOGY	GOAL	ACTIVATE	PRIORITY	DELETE POLICY
0	Any - Any Any	Monitoring	Access WLAN	Any	<input type="checkbox"/>	High	<input type="button" value="Delete"/>
0	Any - Any Any	Monitoring	Core Fixed IP	Any	<input type="checkbox"/>	High	<input type="button" value="Delete"/>
0	Video - QoS Any	Reconfiguration	Access WLAN	Load Balancing	<input type="checkbox"/>	Low	<input type="button" value="Delete"/>
0	Video - QoS Excellent	Reconfiguration	Access WLAN	Load Balancing	<input type="checkbox"/>	High	<input type="button" value="Delete"/>
0	Video - QoS Excellent	Reconfiguration	Core Fixed IP	Load Balancing	<input type="checkbox"/>	High	<input type="button" value="Delete"/>

Reset policies to the last saved

Send the activated policies

Figure 4: Screenshots of Service Definition tabs in the H2N tool

Ontology Concept	Policy Composition	Policy Activation	NEM Conflicts	Database Management	Available NEMs
------------------	--------------------	-------------------	---------------	---------------------	----------------

Policy Composition

Service SLA

Process

[Select Process](#)

Network Domains & Technology

Goal

[Select Goal](#)

[Save](#)
[Clear](#)

WLAN Infrastructure Management

WLAN Infrastructure Management Details

NEM SPECIFICATION NAME: WLAN Infrastructure Management PROVIDER: NKUA VERSION: 1.0.0 RELEASE DATE: Fri May 31 16:19:35 EEST 2013 FEATURES: Features of This NEM provides monitoring information and reconfiguration actions for WLAN devices (Mobile Terminals, Access Points, Network Domain Managers) USER GUIDE URL: http://scan.di.uoa.gr TECHNOLOGY WLAN NETWORK SEGMENT ACCESS POSSIBLE HOSTS Name: Linux OS Version: " Arch." MANAGEABLE ENTITIES Socetris NET5501 LinkSys WRT54GL Ubuntu 10.04 LTS Terminal FUNCTIONALITY FAMILY Resource IS ATOMIC NEM: false IS ATOMIC LOOP: true	AVAILABLE OUTPUTS List of associated APs List of associated terminals per AP Operating System Name Operating System Architecture Operating System Version Java Runtime Environment Version Java Runtime Environment Vendor POSSIBLE ACTIONS Adjust the transmission power of an AP Communicate the priority list of candidate APs to be associated to each mobile terminal Communicate a handover decision to each mobile terminal GOV RULES Name: Policy_00001_WIM Description: User classes for meeting operator SLAs Event: QoS assignment to users Condition: User QoS Class Value: Bronze Name: Policy_00002_WIM Description: Load Balancing action through execution of handover Event: Cell Utilization Condition: CU Value: 0.2
--	--

Figure 5: Screenshots of Service Supervision tabs in the H2N tool

The Service defined through the user interface leads to a policy that can be expressed in a close-to-natural language as follows:

On the Access Part of the network, with specific technology 'WLAN', process of type 'Reconfiguration' should exist, in order to achieve goal 'Load Balancing', taking also into consideration users who consume 'Video' service and have SLA agreement, which reassure 'QoS Excellent'.

This kind of human-readable policy should be further translated into a language that can be understood by the different elements involved in the service provisioning operations. This task is accomplished by the Policy Derivation and Management function.

2.3.2 Coordination

The conflict identification is based on UMF specification in D2.4 [2]. Specifically, it relies on using the instance descriptions to build mathematical graphs plotting all the UMFInformationSpec acquired by NEMs and all the ManagementActionSpecs managed by NEMs. A conflict is detected when a UMFInformationSpec is part of two control loops belonging to two different NEMs. The images below are presenting captures of the conflict identifier GUI without and with conflicts identified.

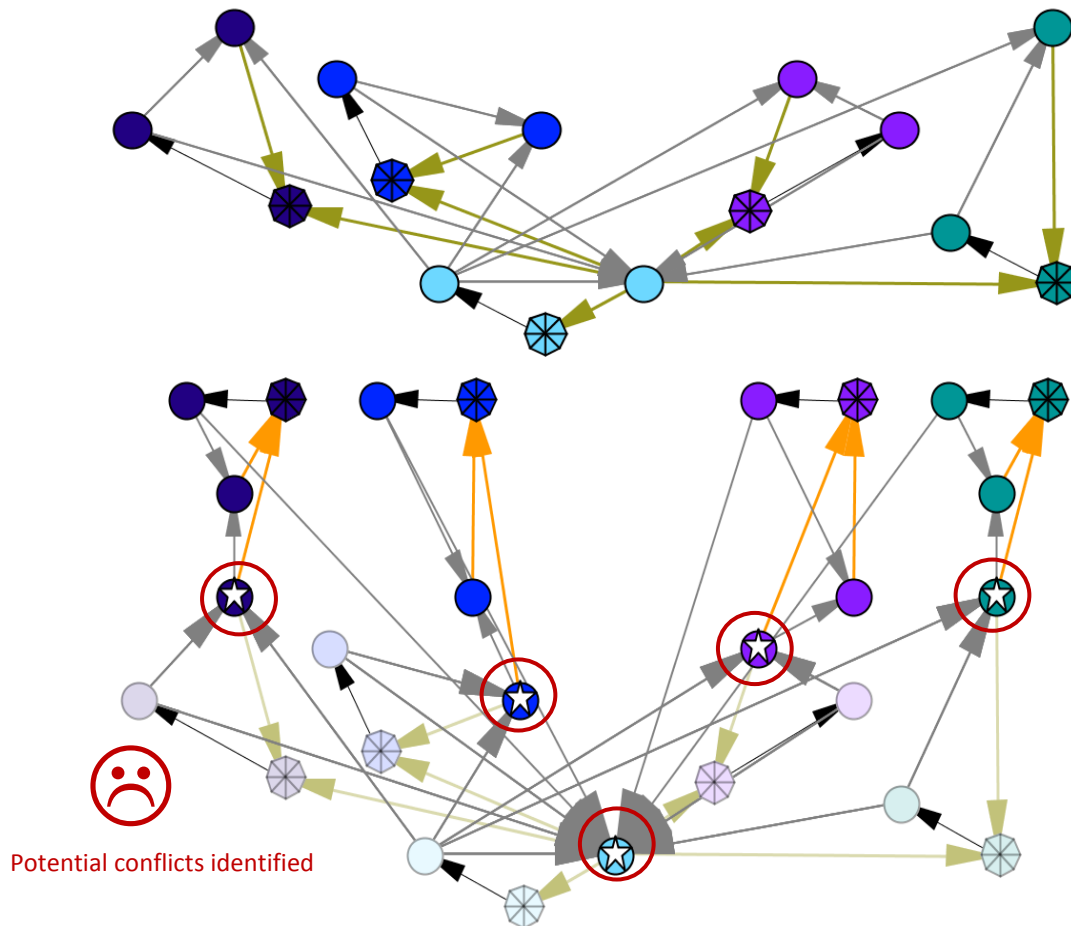


Figure 6: Graphs without (above) and with (below) conflict identified

Section 3.2 includes a detailed explanation of the conflict graphs in the context of a particular, demonstrated use-case. The time separation mechanism that has been implemented imposes different regimes to each of the conflicting NEMs. It should be noted that the NEMs are working at different frequencies when the TimeSeparation mechanism is functioning.

Figure 7 below shows the GUI of the conflictAssigner which instantiated the TimeSeparation mechanism.

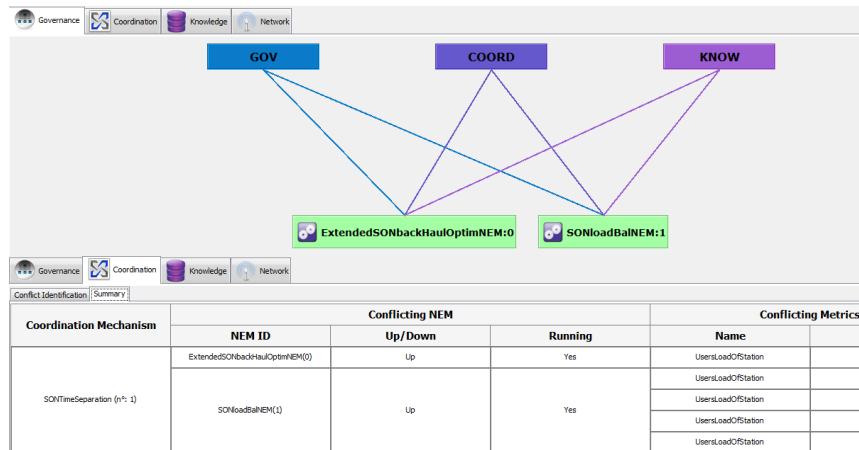


Figure 7: Conflict assigner showing the TimeSeparation assigned

2.3.3 Knowledge

The KNOW block implementation is consistent with the UMF v3.0 specifications documented in D2.4 [2] and its user interface is part of the unified UMF dashboard. The corresponding GUI (Figure 8) shows an overview of the KNOW architecture at the left hand side and real-time measurements regarding its operation at the right hand side. The different measurements can be enabled or disabled through the KNOW configuration file. They include the instantaneous CPU load of KNOW related processes, the memory state required and the total numbers of active flows and numbers of flows per communication method (i.e., pub/sub, push/pull and direct communication).



Figure 8: The Knowledge Block Integrated in the UMF Demo Dashboard

By clicking the output tab, a view of the Knowledge Block console output is enabled (i.e., Figure 9). The details of the communication between the UMF entities (i.e., NEMs or other UMF core blocks) and the KNOW is being displayed through corresponding console messages. A message workflow graphical representation of the same communication can be shown through the Message Log tab. Different workflows show alternative viewpoints

of the KNOW related communication (i.e., Figure 10). The IFEO viewpoint shows the messages exchanged in the Information Flow Establishment and Optimization Function. This workflow is useful for highlighting the information/knowledge flow negotiation activity of the Knowledge block. The NEMS2KNOW show the messages exchanged between the UMF entities (i.e., either core services or NEMs) and the Knowledge block. It is actually considering KNOW as a black box. The KNOW functions workflow shows the messages exchanged between the different KNOW functions. So, the KNOW activities are totally transparent to the administrator.

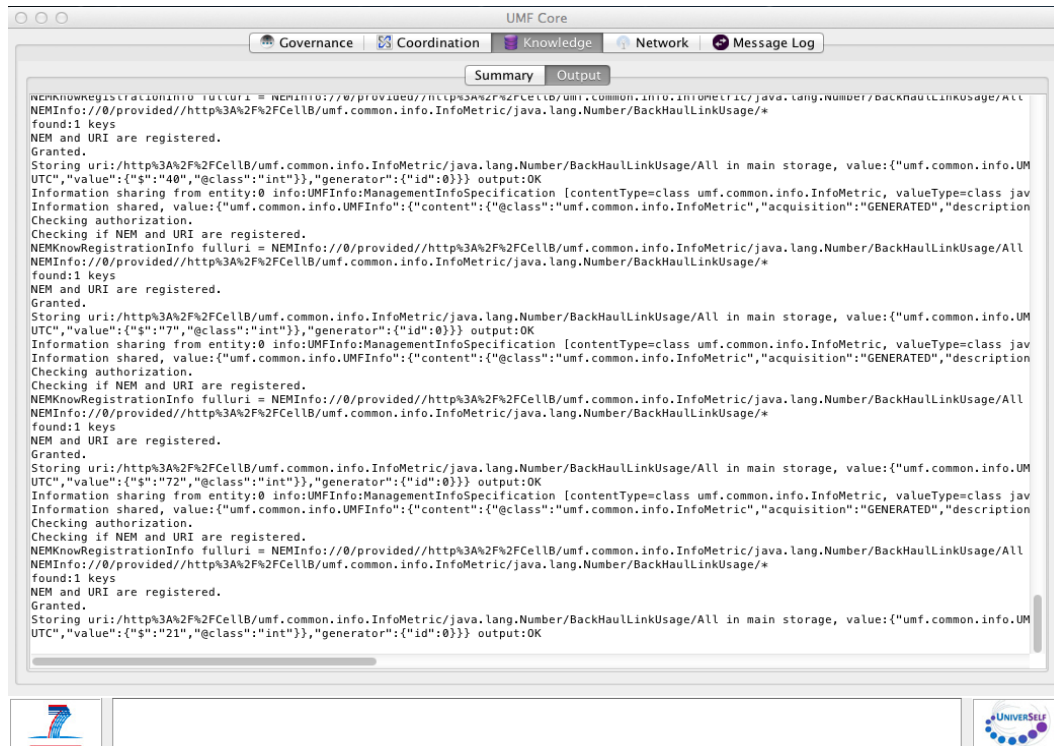


Figure 9: The Knowledge Block Console Output

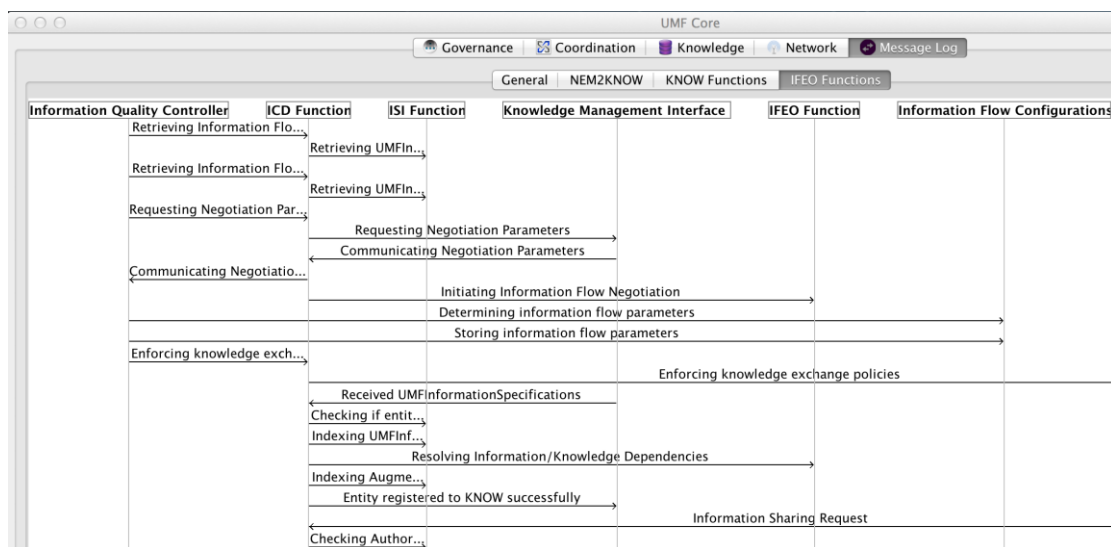


Figure 10: The Knowledge Block Related Workflow Diagrams

3 Integrated Demonstration Scenarios

Based on the above-described UMF core functionality, three, almost orthogonal, scenarios are sketched so as to highlight different UMF aspects with respect to the overall project objectives. Each scenario exploits the same UMF core blocks' backhaul functionality; however, in each scenario different NEMs are deployed, in different and heterogeneous network domains and for different scopes (e.g. QoS preservation, SON, network optimization, conflict resolution etc) to highlight the “unified” nature of UMF. In the following subsections, the three integrated scenarios will be presented in more detail.

3.1 A Unified Framework for QoS and SLA-aware multi-domain self-management

This scenario targets to validate UMF functionality over a multi-domain, multi-technology data plane, respecting also QoS and SLA constraints [25]. Specifically, the network topology under consideration consists of a real WLAN access segment and an SdN-based core network topology that emulates an IP core network and is comprised of several virtual routers. Deployment and orchestration of different NEMs is achieved through UMF. A two-phase scenario is presented where reconfiguration activities take place so as to achieve QoS constraints with or without SLA criteria.

3.1.1 Multi-domain testbed description

A heterogeneous multi-domain topology is considered where mobile users consume video services, respecting QoS and SLA constraints. In the considered topology a WLAN access network is deployed so as to provide wireless users connectivity. More specifically, two soekris based WiFi access points, enriched with monitoring capabilities are responsible for serving three mobile users. On the other hand, an SdN-based network topology is deployed as the core part of the topology. Seven virtual routers are responsible for forwarding UDP traffic. Soekris 2 is considered as an edge node that interconnects core and access parts of the network. A video server provides content to the end mobile users.

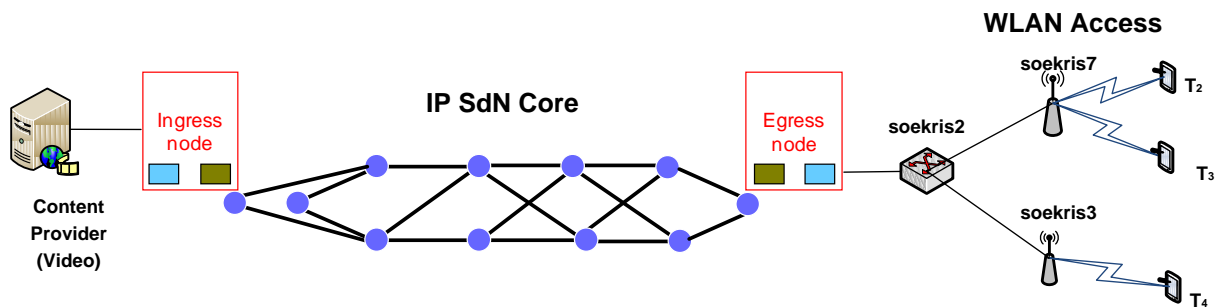


Figure 11: Heterogeneous topology.

Soekris devices are responsible both for providing network connectivity to the end users and monitor in a periodic manner their status and load oriented metrics. An extensive list of monitoring information is available in Table 1.

Information/Metric Name	Description
Basestation ID	The ssid of the WiFi Access Point
Channel	The 802.11 channel the device transmits
TxPower	The transmission power of the wireless antenna
Cell Utilization	The available bandwidth
List of associated Terminals	An extensive list of the associated terminals and their scan results

Table 1 Soekris WiFi list of monitoring metrics

Mobile Terminals are also enriched with monitoring and reconfiguration capabilities. An extensive list of the pieces of information available from terminals is provided in Table 2.

Information/Metric Name	Description
Terminal ID	A unique identifier of terminal
IP address	The IP address provided by the soekris the terminal is associated to
Basestation ID	The ESSID of the WiFi the terminal is associated to
Monitoring ESSID	The ESSID of WiFi Access Points the terminal senses
Monitoring RSSI	The received signal strength of the neighbouring WiFi Access Points
SLA	The user class the terminal belongs to, based on the SLA agreement

Table 2: Mobile Terminal list of monitoring metrics

Finally, a core SDN topology has been deployed comprising of seven routers, so as to set up the core part of the heterogeneous topology. More specifically, one ingress, one egress and five forwarding routers have been set up with their corresponding connections (see Figure 20). An extensive and detailed description regarding the set up process of the topology and the creation of new UDP service paths is available in section 3.3.2.1

3.1.2 Heterogeneous NEM deployment

To address the above described constraints, 4 different NEMs are deployed in order to orchestrate the data plane. Two NEMs are instantiated in the access segment, namely, WLAN Infrastructure Management NEM and Wireless access Load Balancing NEM. For the core segment, Virtual Infrastructure Management NEM and Core Load Balancing NEM are instantiated as well. Figure 12 visualizes UMF and NEMs deployment.

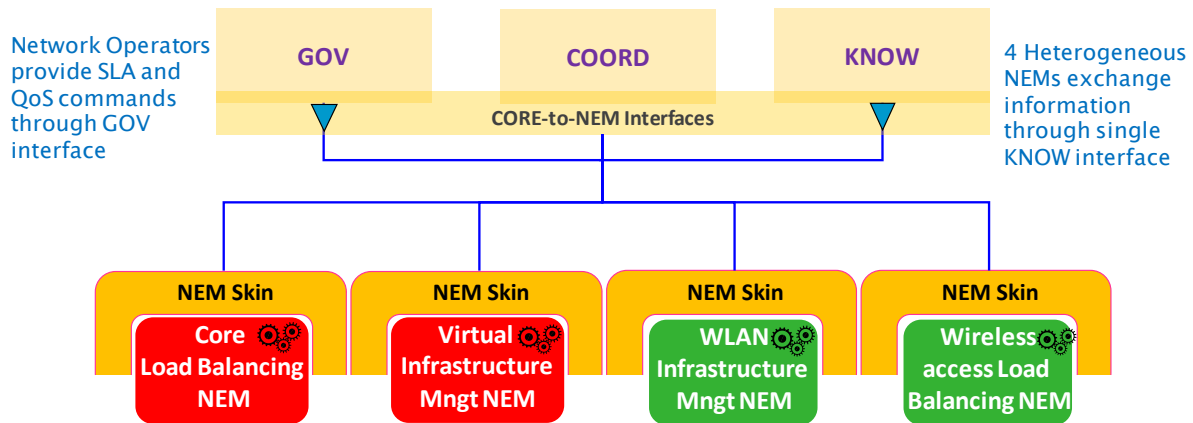


Figure 12: Integrated scenario 1 - UMF and NEMs.

In a higher technical granularity, NEMs' functionality is presented above:

1. WLAN Infrastructure Management NEM
 - Monitoring of the topology (#APs, #MTs), network parameters (CU, RSS, Noise, Link Quality, Tx),
 - NEs' Reconfiguration (Txpower update, MT reallocation) dictated by Wireless access load balancing NEM through KNOW.
2. Wireless access Load Balancing NEM
 - Derivation of complex load parameters (i.e. Device Status based on CU and PL) for load balancing,
 - UMF KNOW update with handover decisions to be executed by WLAN Infrastructure Management NEM.
3. Virtual Infrastructure Management NEM

- Dynamic instantiation and deployment of SdN topology,
 - Reconfiguration of virtual routers (Ingress/Egress path establishment and adaptation, forwarding, re-routing).
4. Core Load Balancing NEM
- Monitor service flows,
 - UMF KNOW update with link activation/deactivation commands to be executed by Virtual Infrastructure Management NEM.

The above-presented UMF-driven NEM capacities are able to provide:

1. Network Operator's business goals realization (i.e. QoS/SLA constraints).
2. Joint core and access network load balancing accounting network and service parameters,
 - UMF KNOW is the connection point between core and access providing global network and service view by aggregating information from all the NEMs.
 - The executed load balancing actions are user re-allocation for the access segment and flow/path re-routing in the access.
 - Optimal load distribution and CCO is achieved accounting network and service parameters (CU, RSS, RTT, BW), SLAs (User Classes).
3. Automated triggering and execution of load balancing actions in both core and access.
4. On demand deployment and adaptation of SdN nodes across the core network.

The UMF enhanced data plane is presented in Figure 13.

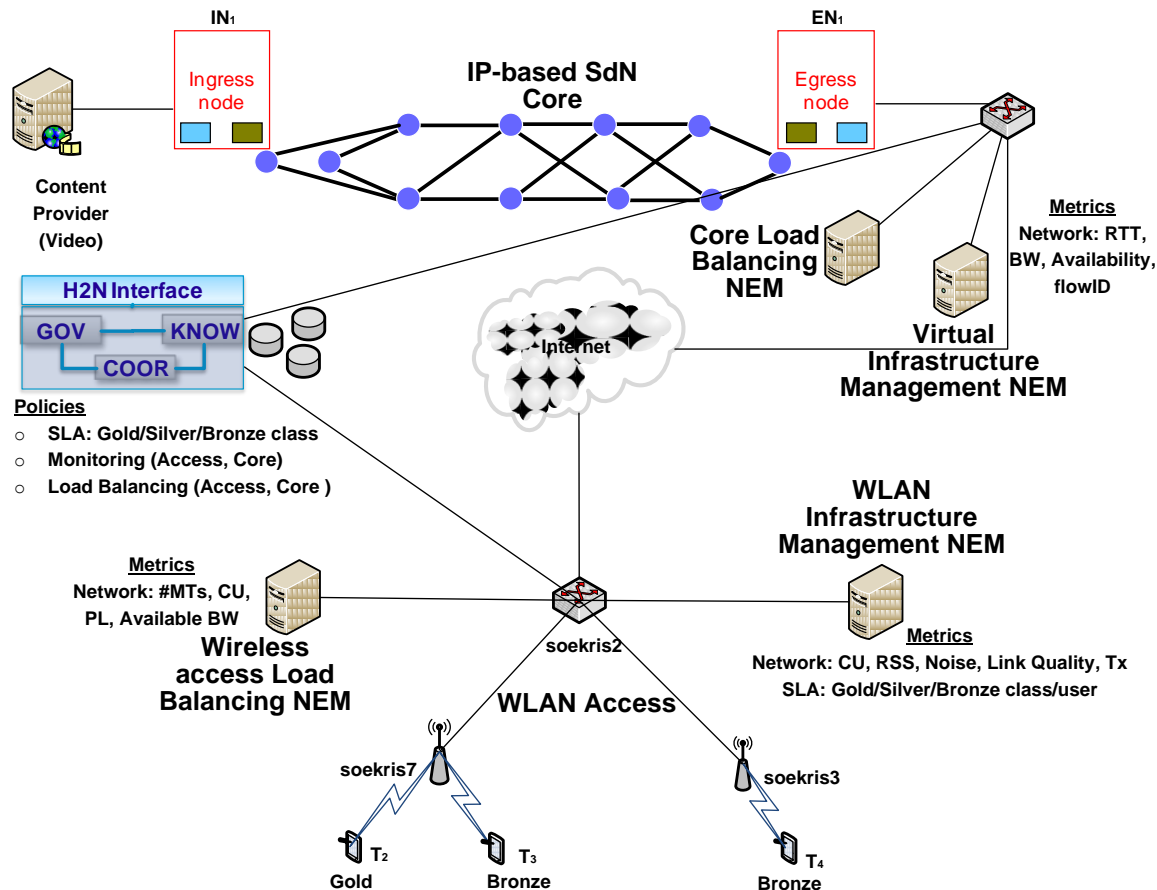


Figure 13: Integrated scenario 1 – UMF enhanced data plane.

3.1.3 Scenario Storyline

In this section a more detailed storyline description of the scenario will be given. Initially, UMF is deployed while WLAN Infrastructure Management (WIM) and Virtual Infrastructure Management (VIM) NEMs are instantiated and deployed for providing monitoring information of the access and core network respectively. No load is injected in the network, thus no need for reconfigurations is raised.

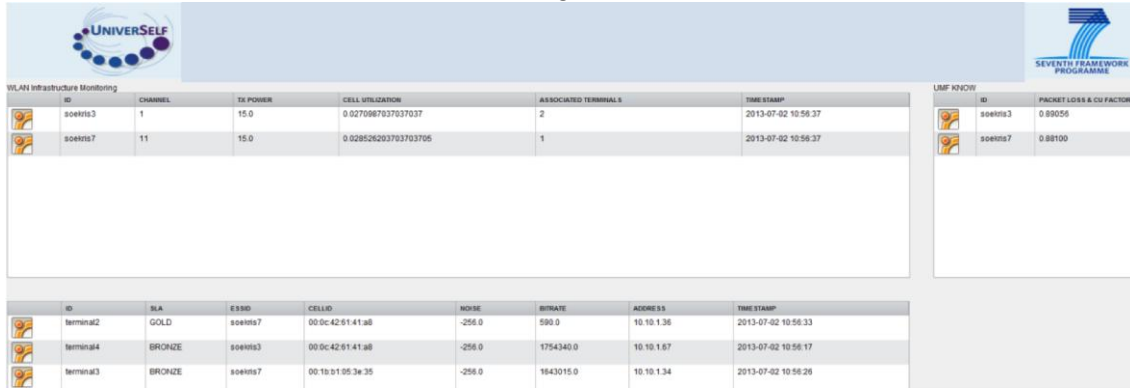


Figure 14 presents access network status, as monitored by WIM NEM.

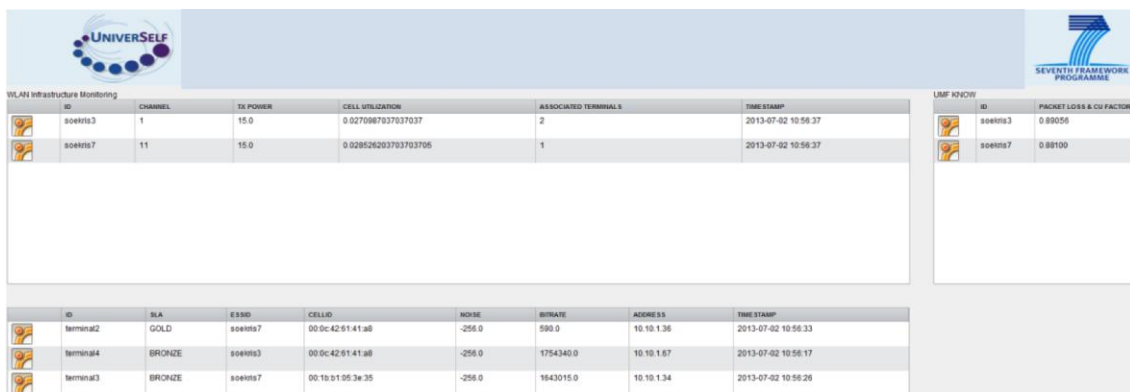


Figure 14: Integrated scenario 1 – Initial access network status – No Load.

Injecting some load in the network, by streaming a video service to the mobile terminals, the load is increased and network operator enables policies for load balancing in the access and core network. Those policies are dictating the deployment of 2 NEMs, namely, Wireless access Load Balancing (WLB) NEM and Core Load Balancing NEM. The former NEM produces a combined metric called Device Status by gathering network measurements from UMF KNOW, as produced by WIM NEM. Once the device status exceeds a predefined level, indicated in the policy coming from UMF Governance, then a reconfiguration action (i.e. a mobile terminal reallocation) is executed by WLB NEM.

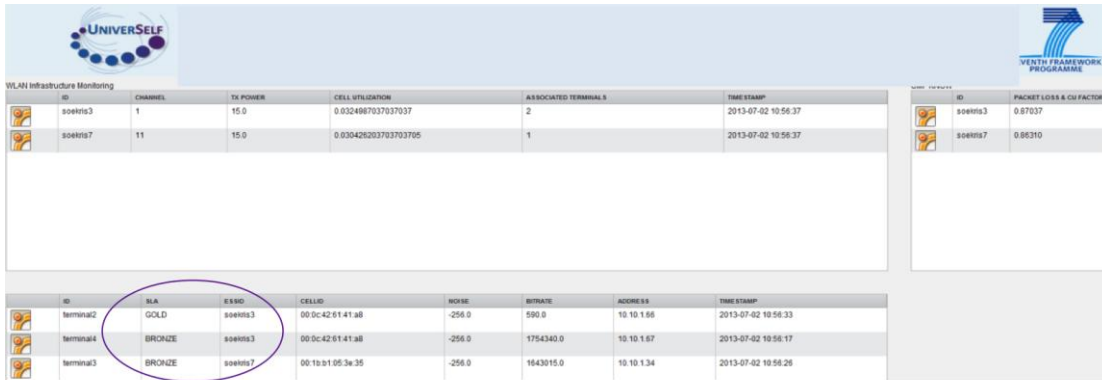
Policy Activation

Activate/Deactivate created policies

ID	SLA	PROCESS	NETWORK DOMAINS & TECHNOLOGY	GOAL	ACTIVATE	PRIORITY	DELETE POLIC
0	Video - QoS Any	Reconfiguration	Access WLAN	Load Balancing	<input checked="" type="checkbox"/>	High	Delete
0	Video - QoS Excellent	Reconfiguration	Access WLAN	Load Balancing	<input type="checkbox"/>	Low	Delete
0	Any - Any Any	Reconfiguration	Core Fixed IP	Load Balancing	<input checked="" type="checkbox"/>	High	Delete
0	Any - Any Any	Monitoring	Access WLAN	Any	<input checked="" type="checkbox"/>	High	Delete
0	Any - Any Any	Monitoring	Core Fixed IP	Any	<input checked="" type="checkbox"/>	High	Delete
0	Any - Any Any	Reconfiguration	Access WLAN	Emergency	<input type="checkbox"/>	Low	Delete

Figure 15: Integrated scenario 1 – Policy activation for load balancing.

Figure 15 visualises UMF Governance H2N interface for policy activation. Currently, monitoring and reconfiguration policies are activated for both access and core segments, however no SLA constraints are being put to address video QoS requirements for each user class. This means that it is highly possible to execute a mobile terminal re-allocation that belongs to a GOLD class user. Indeed, based on Device Status metric, the GOLD user is re-allocated to the neighbouring AP, as shown in Figure 16.



The figure shows the UMF Governance H2N interface. It includes a 'WLAN Infrastructure Monitoring' table and a 'Device Status' table. The 'Device Status' table has a row for 'terminal2' with SLA 'GOLD' circled in red.

ID	SLA	ESSID	CELLID	NOISE	BITRATE	ADDRESS	TIME STAMP
terminal2	GOLD	soeris3	00:0c:42:61:41:a8	-256.0	590.0	10.10.1.66	2013-07-02 10:56:33
terminal4	BRONZE	soeris3	00:0c:42:61:41:a8	-256.0	1754340.0	10.10.1.67	2013-07-02 10:56:17
terminal3	BRONZE	soeris7	00:19:11:05:3a:35	-256.0	1643015.0	10.10.1.34	2013-07-02 10:56:26

Figure 16: Integrated scenario 1 – User re-allocation according to network metrics.

This decision, taken in the access segment, affects also the core network since the service flow should be redirected to the neighbouring AP so as to preserve video and avoid service interruption. The Core Load Balancing (CLB) NEM automatically executes this flow/path reconfiguration, once the WLB NEM specifies a handover decision. The egress node of the core segment updated the IP address of the mobile terminal. This is feasible since all the NEMs are able to communicate each other through UMF Knowledge block. Figure 17 visualizes CLB reconfiguration results.

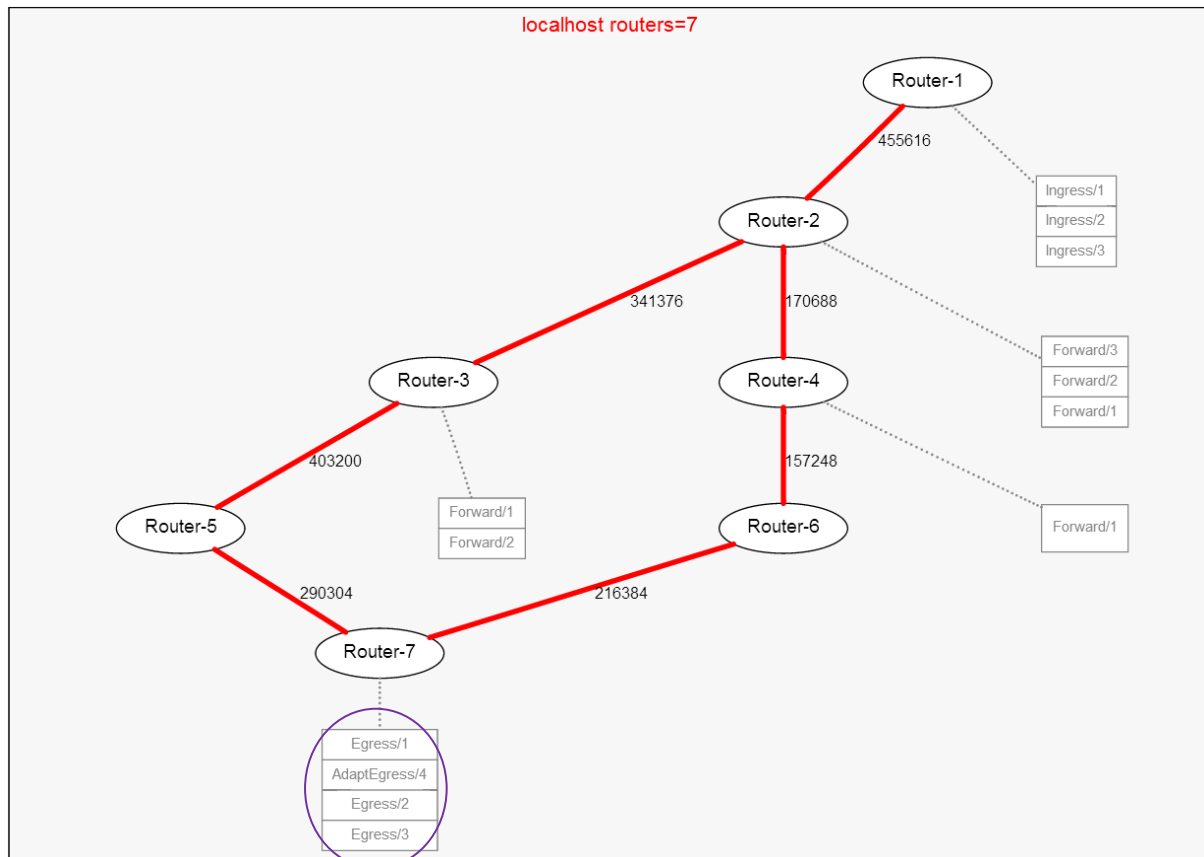


Figure 17: Integrated scenario 1 – Egress path adaptation in the core network together with the first access network handover.

The above-taken decision is not the optimal one since within a real network environment end users are assigned with SLAs which reflect the QoS that they expect for their services from the network operator while in parallel the cost of this QoS provision. This means that SLAs should be accounted in the big figure; network operator can easily enable this option using UMF Governance H2N interface. By composing a new policy (Figure 18), which assigns user classes to end-users for a particular service, WLB NEM is able to update its decision-making.

Policy Activation

Activate/Deactivate created policies

ID	SLA	PROCESS	NETWORK DOMAINS & TECHNOLOGY	GOAL	ACTIVATE	PRIORITY	DELETE POLIC
0	Video - QoS Any	Reconfiguration	Access WLAN	Load Balancing	<input type="checkbox"/>	Low	Delete
0	Video - QoS Excellent	Reconfiguration	Access WLAN	Load Balancing	<input checked="" type="checkbox"/>	High	Delete
0	Any - Any Any	Reconfiguration	Core Fixed IP	Load Balancing	<input checked="" type="checkbox"/>	High	Delete
0	Any - Any Any	Monitoring	Access WLAN	Any	<input checked="" type="checkbox"/>	High	Delete
0	Any - Any Any	Monitoring	Core Fixed IP	Any	<input checked="" type="checkbox"/>	High	Delete
0	Any - Any Any	Reconfiguration	Access WLAN	Emergency	<input type="checkbox"/>	Low	Delete

Figure 18: Integrated scenario 1 – Policy activation for SLA-aware load balancing.

In particular, upon a user re-allocation event, WLB NEM will decide about the optimal handover considering both network and SLA parameters. Therefore, in a similar high load event, the new re-allocation results are showcased in Figure 19. The updated decision performs Bronze user re-allocation respecting SLAs and improving the overall Device Status of the access network.

ID	CHANNEL	TX POWER	CELL UTILIZATION	ASSOCIATED TERMINALS	TIME STAMP
sco0003	1	15.0	0.54021448740740742	1	2013-07-02 11:00:54
sco0007	11	15.0	0.15130485185185185	2	2013-07-02 11:00:54

ID	SLA	ESMD	CELL ID	MWDR	BYRATE	ADDRESS	TIME STAMP
terminal02	GOLD	sco0003	00:0C:42:61:41:A8	-256.0	7578978.0	10.10.1.86	2013-07-02 11:00:45
terminal03	BRONZE	sco0007	00:10:01:05:34:35	-256.0	8168488.0	10.10.1.34	2013-07-02 11:00:47
terminal04	BRONZE	sco0007	00:10:01:05:34:35	-256.0	864.0	10.10.1.36	2013-07-02 11:00:39

ID	PACKET LOSS & CPU FACTOR
sco0003	0.9454
sco0007	-0.9547

Figure 19: Integrated scenario 1 – User re-allocation according to network and SLA metrics.

Besides the right decision in the access network, a respective reconfiguration takes place in the core by adapting egress service flow path as shown in Figure 20.

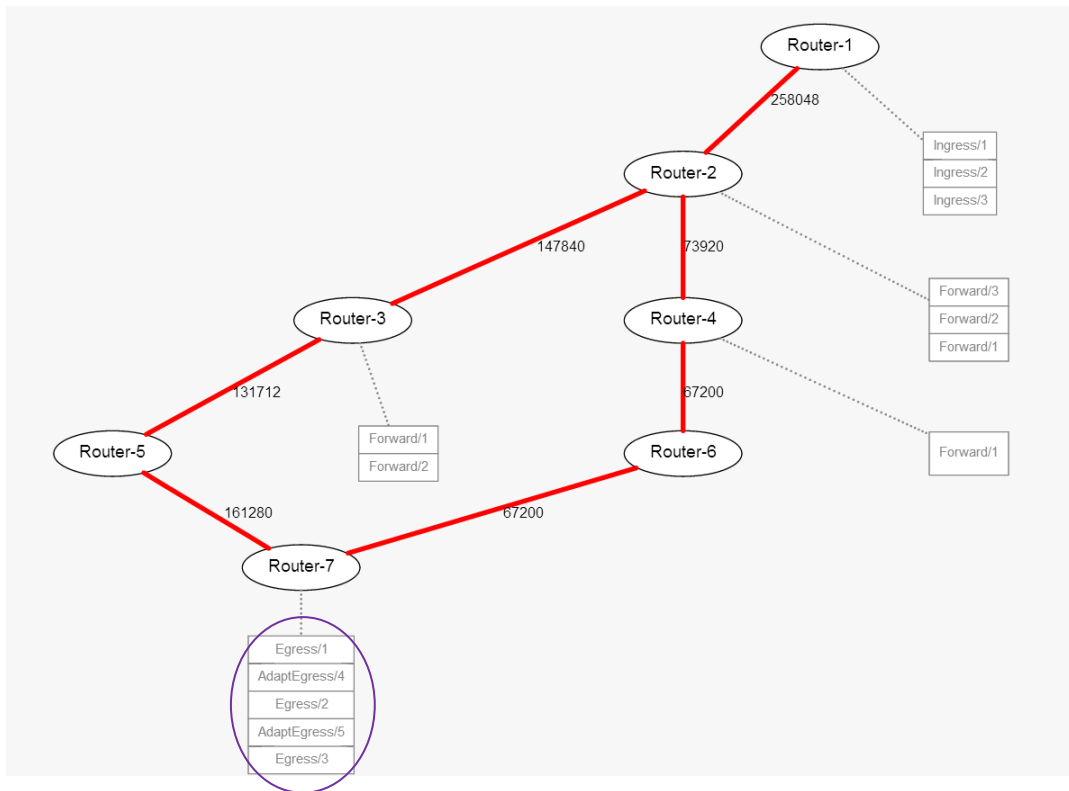


Figure 20: Integrated scenario 1 – Egress path adaptation in the core network together with the second access network handover.

A live demonstration of the scenario was part of the UniverSelf demo booth presented in Future Network and Mobile Summit 2013[9] in Lisbon. Figure 21 presents the demo booth setup of this scenario.

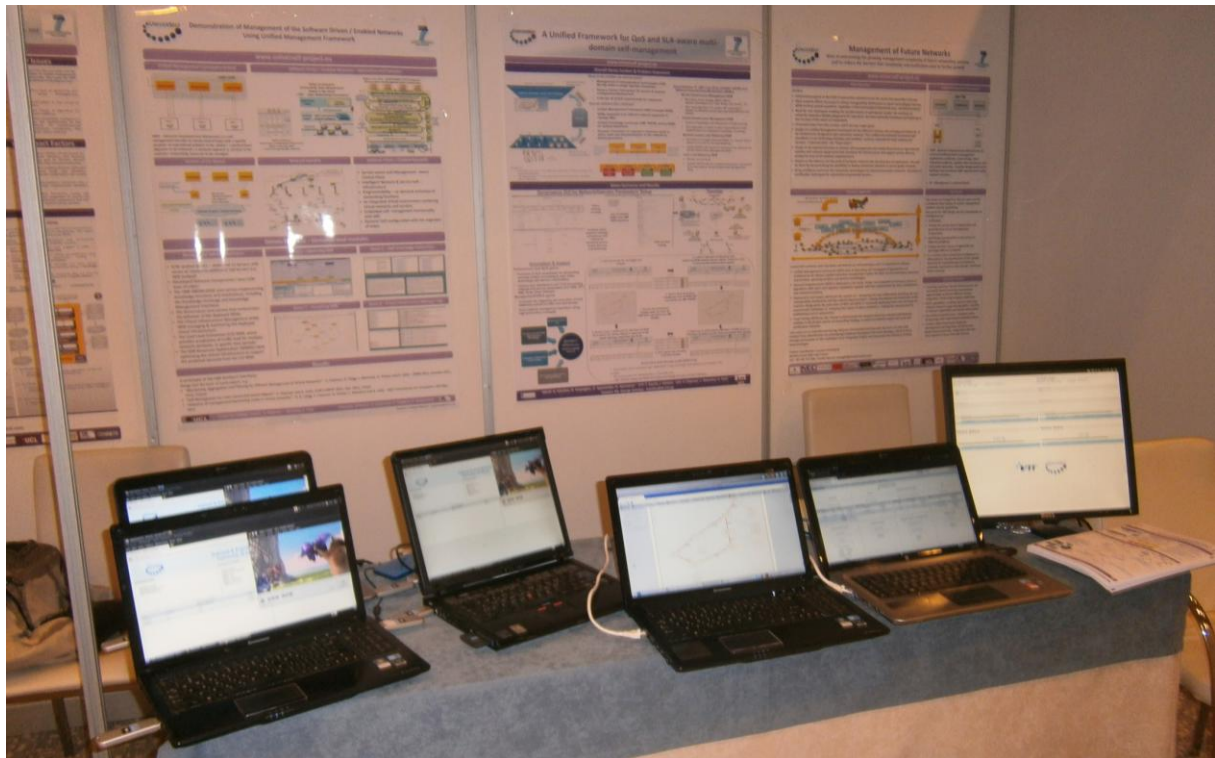


Figure 21: Integrated scenario 1 –FUNEMS 2013 booth.

3.1.4 Summary

Concluding this section, the afore-presented scenario validates the performance of UMF deployment in a heterogeneous, multi-domain network where the service view is also supported. This is achieved through the unified, UMF Governance-driven orchestration of different NEMs, able to operate in the different network segments (i.e. access and core) and technologies (i.e. IP-based SdN and Wi-Fi) and intercommunicate using a single interface (through NEM Skin knowledge exchange interface). On the one hand network operator is able to infuse its own high level policies to manage the network in a highly abstract way, including also SLA requirements while on the other hand NEMs are able to collect and exploit network and service metrics. Combining this information, network management decisions are optimized since they capture a multi-variate knowledge of the network ecosystem in a very effective way. Therefore, joint load balancing can be easily performed since it is assisted by the distributed knowledge aggregation of UMF Knowledge block.

3.2 Conflict-free Coordination of SON NEMs

3.2.1 Scenario Storyline

In this scenario, conflict-identification and -resolution function of the Coordination block of UMF are highlighted in an LTE HetNet with Self-Organizing Network (SON) functions, which have been deployed as NEMs.



Figure 22: Typical situation of SON functions (NEMs) not being aware of other affecting functions' existence and operation

The LTE testbed over which the demonstration takes place has been simulated in Matlab and visualized in Java. In this particular instantiation, it consists of 1 LTE macro cell (see *CellB* in Figure 23) and 4 small cells (Femtos, see *FemtoB*.{1-4} in Figure 23) backhauled through the Macro cell using wireless links (in-band relaying). Omnidirectional antennas are modelled on a cell radius of 2km, a Rayleigh fast-fading model and 46dBm/30dBm transmit power for macro/small cell antennas respectively. OFDMA is the access technology used, while 10 resource blocks of 180kHz each are assumed. Users (see shades of blue in Figure 23) arrive randomly in the cells following a spatial Poisson process with, initially fixed and equal arrival rates for each Femto. Each user is assumed to download a 10MB file and then leave the network.

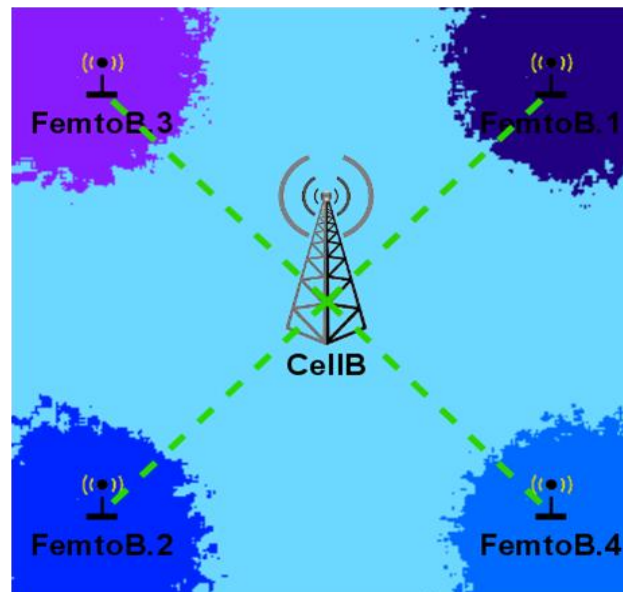


Figure 23: Simulated LTE SON used on the SON Coordination scenario

The two SON functions (NEMs) considered, implement self-optimization functions using a hierarchical conflict free setup. SON 1 (Load Balancing -LB in short- NEM) adjusts relays' coverage to offload the right amount of eNodeB traffic, and SON 2 (BackhaulOptimization -BHO in short- NEM) balances radio resources between backhaul and station-to-mobile links.

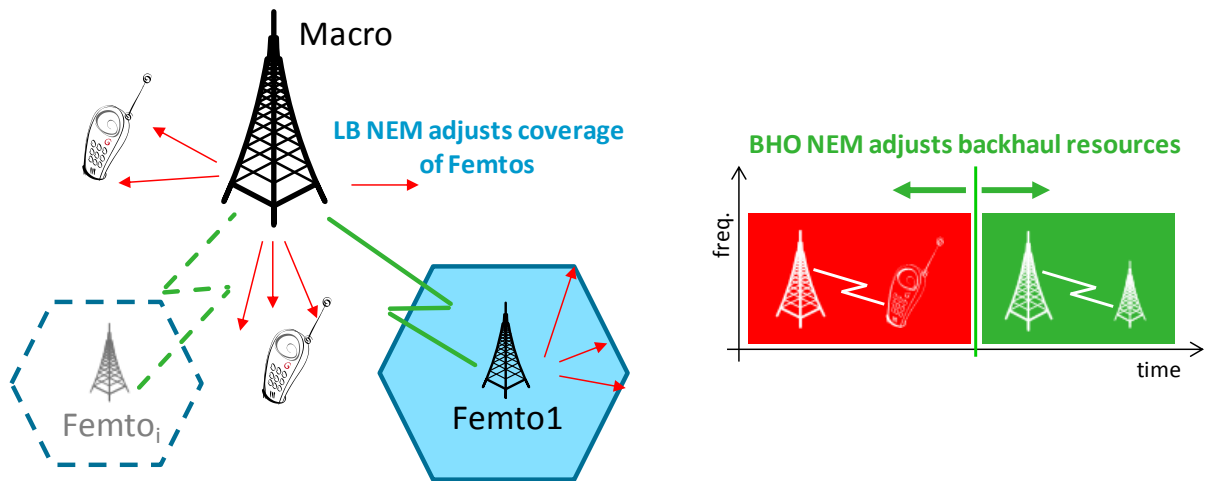


Figure 24: Simplified visualization of the functionality of the 2 two NEMs

One instance per NEM is created and deployed over the simulated RAN through the UMF dashboard (see Figure 25).

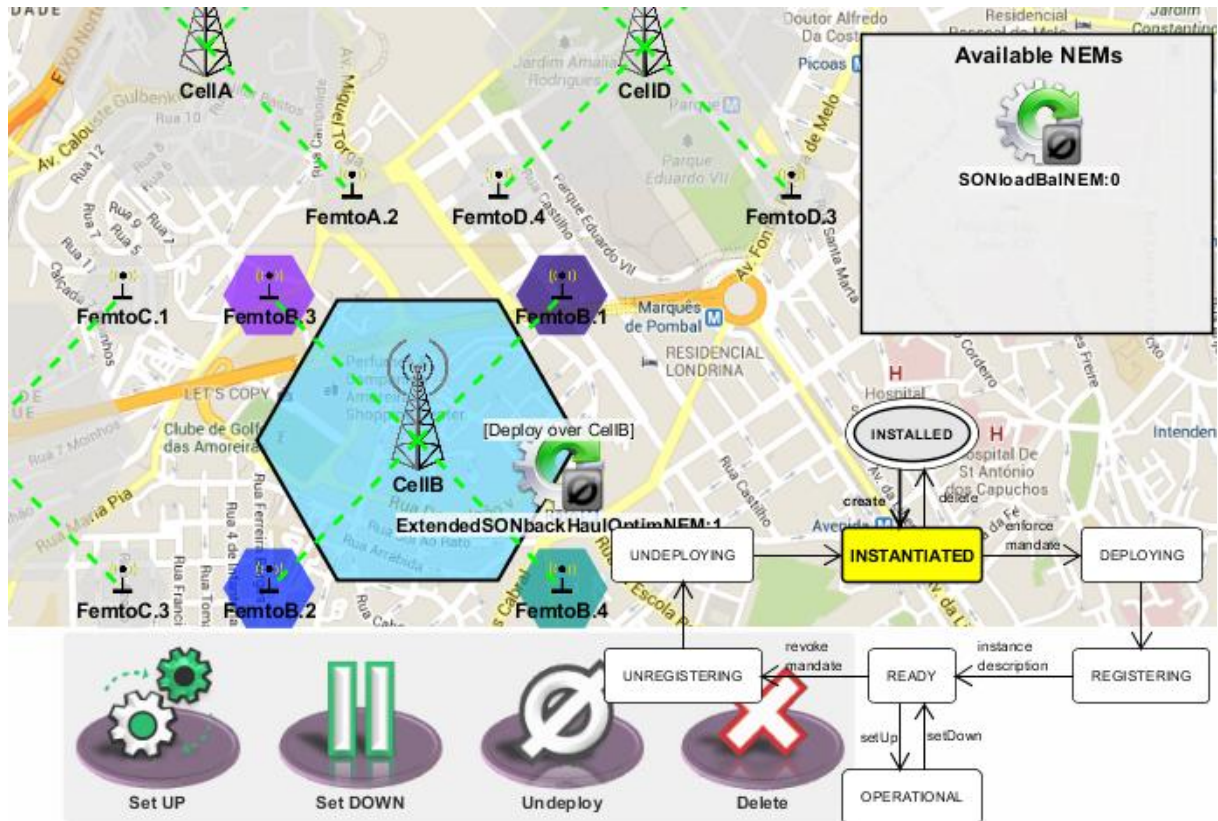


Figure 25: The LTE RAN view of the UMF dashboard just prior to the deployment of the NEMs

At that time, the users in the network are uniformly spread and there is no load balancing to be performed by the LB NEM. Prior to turning on (UP-state) these NEMs, Coordination receives their registration request with an enclosed instance description (as per the UMF specifications) for each, containing the elements to be managed (CellB in this case), any UMF information specifications to be produced and consumed as well as the UMF actions to be taken (their specifications). Conflict identification then relies on such instance descriptions to build mathematical graphs plotting all the specifications acquired by the NEMs and any inter-relations. A potential conflict is detected when aUMInformationSpec is part of two control loops belonging to two different

NEMs. Figures below depict subsets of such a conflict graph produced by the Coordination block and visualized on the H2N interface, prior (Figure 26) and after (Figure 27) potential conflicts are identified.

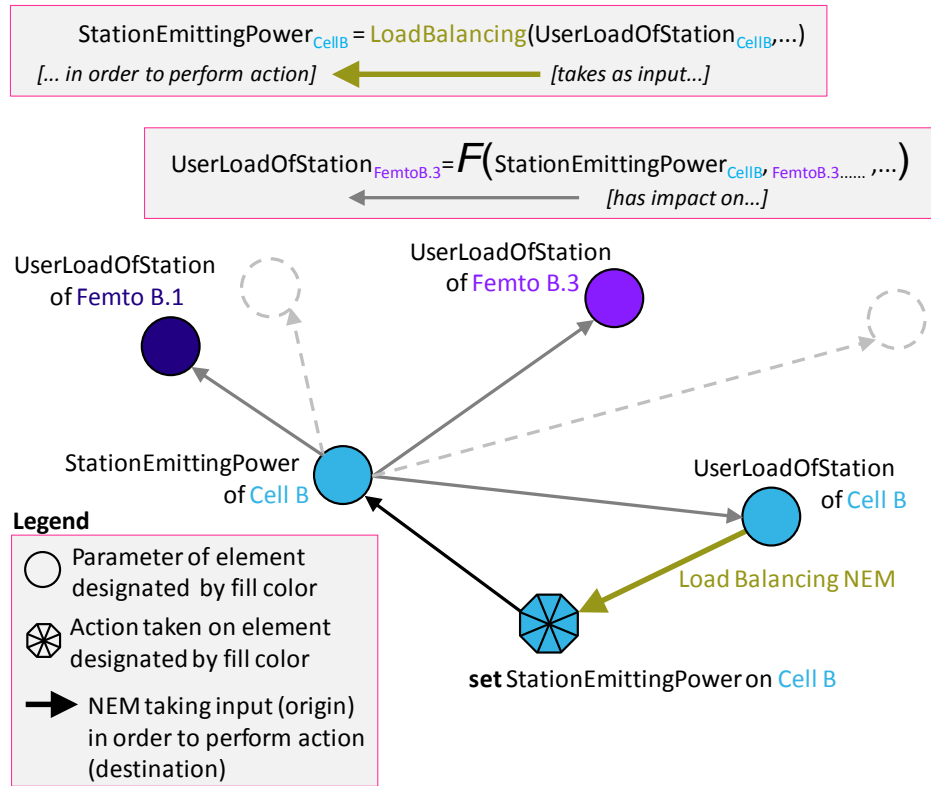


Figure 26: Conflict graph built by Coordination block during the deployment of a Load Balancing SON NEM

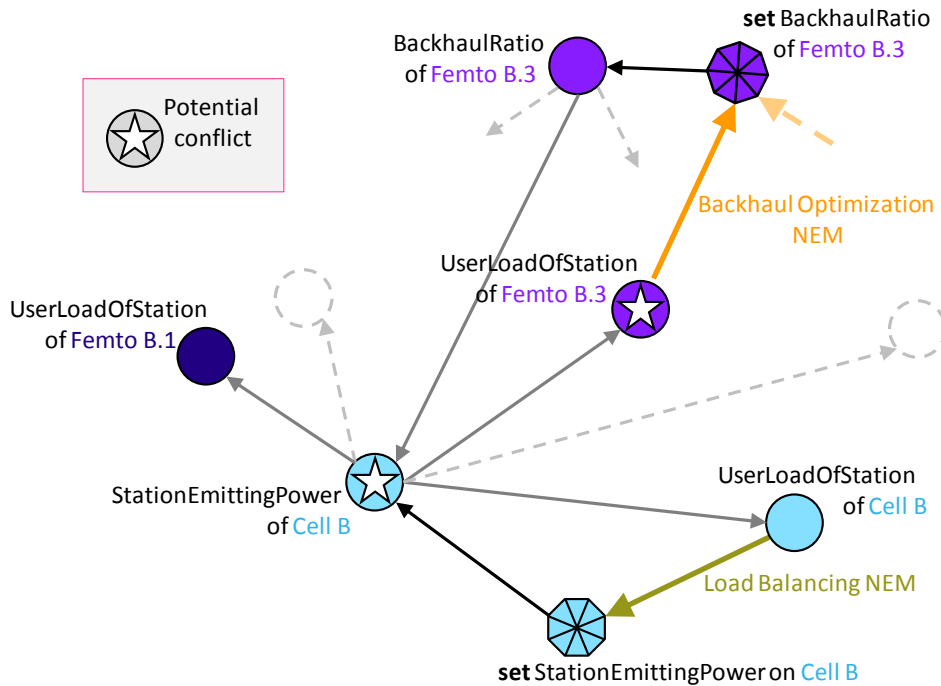


Figure 27: Conflict graph further populated by UMF during the deployment of an additional Backhaul Optimization SON NEM

Colored arrows denote "takes as input" and greyscale arrows, denote "has impact on" associations (derived from the UMF ontology) while the conflicting metrics are denoted with a star. In particular, during the deployment of the BHO NEM (see Figure 27) that takes as input a metric that is being actively modified by the LB NEM, the system identifies a potential conflict/instability during their concurrent execution.

In this particular, demonstrated scenario, potential conflicts are identified, since BHO NEM is taking as input the user load of stations (UserLoadOfStation) in order to decide and take action while at the same time the LB NEM's actions (SetStationEmmittingPower) are highly impacting the very same parameter (due to coverage change; which is knowledge derived from the UMF ontology) and driving BHO to decisions and adjustments that are only temporarily correct, possibly contradicting or non-convergent in the long-term. Therefore, the network is driven to an unstable state unless coordination intervenes.

Therefore, UMF Coordination enforces NEM operation separated in time (see also paragraph 2.3.2) with the correct scales: short and long time scales for NEM 1 and NEM 2 respectively to guarantee convergence and stability. Such paces are applied through corresponding UMF Regimes instructing NEMs to operate continuously but with different time intervals between each iteration through their MAPE loops.

Finally, for demonstration purposes, after the NEMs are turned on (UP-state), and coordinated, the user arrival rates are manually changed to unbalance the traffic and demonstrate the effects of both the NEMs on their own, but also of the Coordination mechanism being applied on them.

3.2.2 Results

Figure 28 and Figure 29 are snapshots of the collected KPIs for both the non-coordinated and the coordinated versions of the very same deployment scenario, respectively. The lack of coordination is detrimental to the functioning of these two NEMs. As shown in Figure 28, the user experience is bad as the average file transfer time gets up to 25 sec, while with coordination set the file transfer time stays below 4 sec for the same traffic variation scenario.

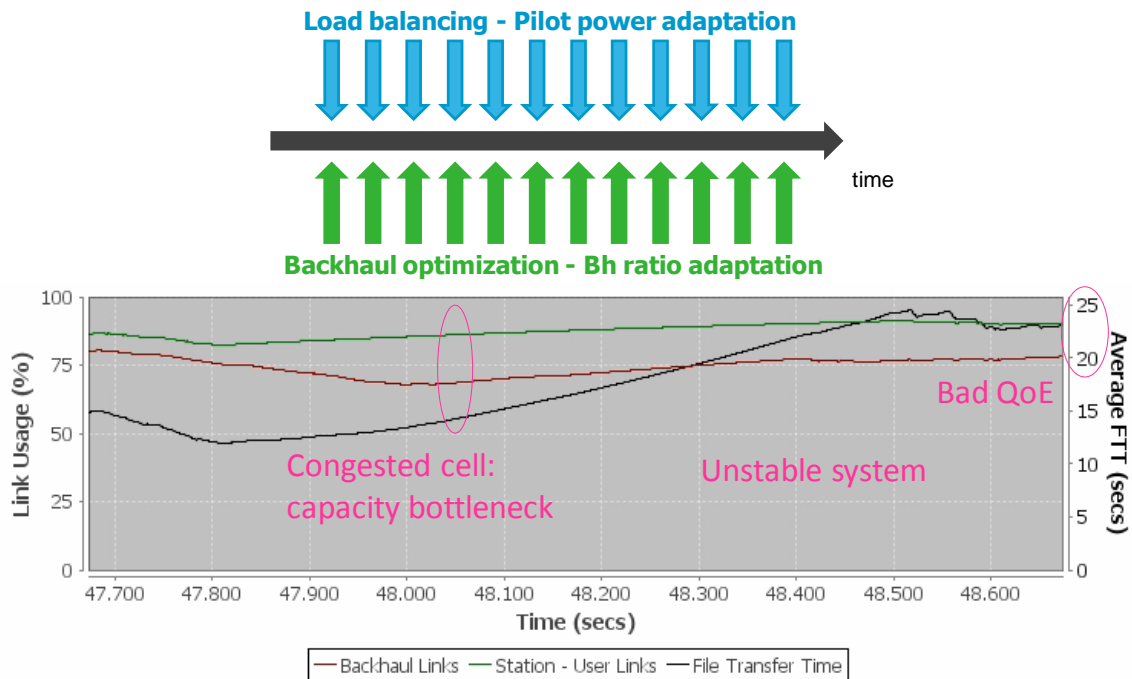


Figure 28: Snapshot of the KPIs when the 2 SON NEMs are not being coordinated (operating in the same pace)

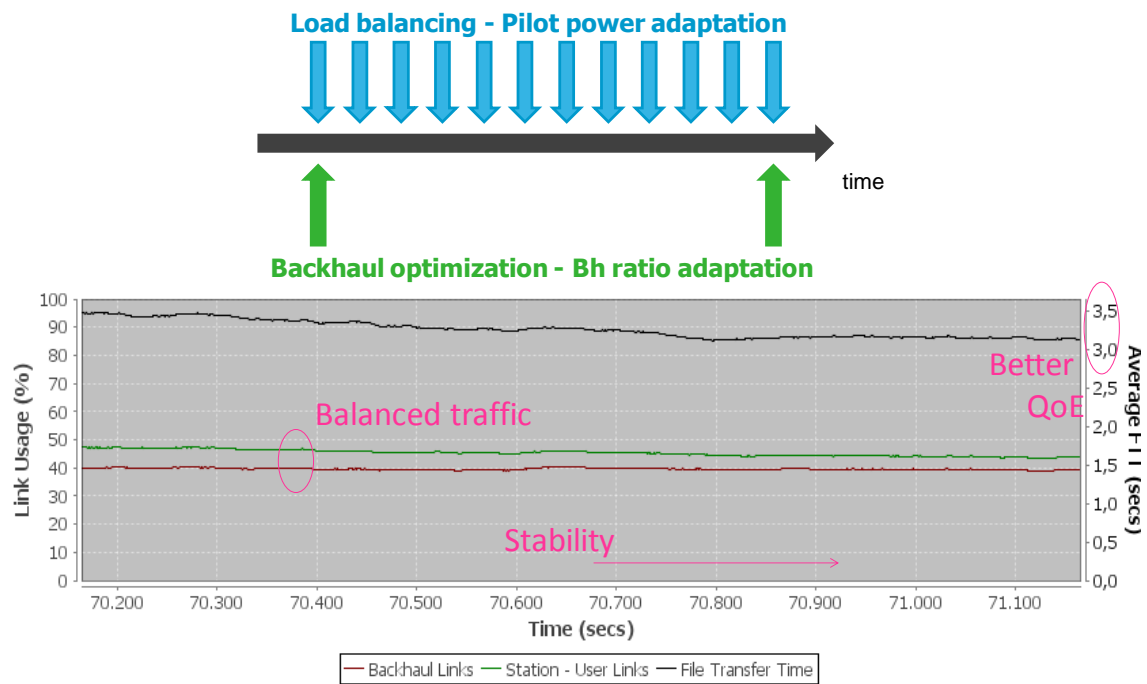


Figure 29: Snapshot of the KPIs of the very same SON NEM combination being coordinated using Time-Separation (slow/fast paces being assigned)

3.2.3 Summary

UniverSelf has embodied world-class research to demonstrate the applicability of adaptable generic solutions for the coordination of SON functions. UMF coordination can allow to fully benefiting from the SON technology by enforcing conflict free operation. At the same time it provides the operator (via Governance) with the means to manage autonomic processes. The successful operation of Coordination is a main enabler for the large scale deployment of SON technology.

3.3 UMF Management of Software Defined Networks

The following presents the developed Very Lightweight Software Driven Network and Service Platform (VLSP) testbed and its use in the UMF management of Software Defined Networks (SDN) [26]. VLSP is planned for release as open-source software under the LGPL licence.

3.3.1 VLSP Overview

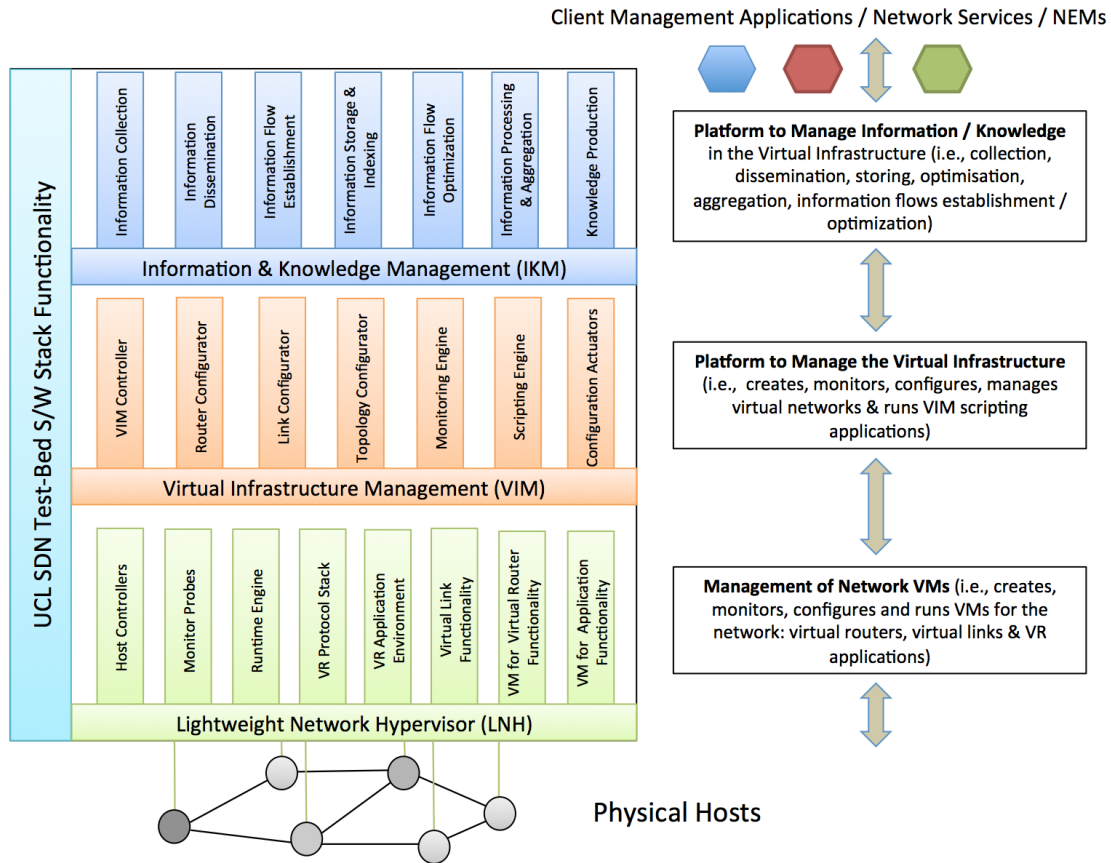


Figure 30: Overview of VLSP Test-bed Software Stack

The Very Lightweight Software Driven Network and Services Platform (VLSP) test-bed includes a new lightweight network hypervisor, a novel infrastructure for the management and manipulation of virtual networks on top of the hypervisor and facilities for handling information / knowledge in the SDN environment.

The VLSP software stack consists of three layers:

- the Lightweight Network Hypervisor,
- the Virtual Infrastructure Management, and
- the information / knowledge management components.

An architectural overview of the software stack is shown in Figure 30. The VLSP test-bed software consists of over 700 java classes and more than 100 k-lines of code. In our experiments with VLSP, we have executed over 100 virtual routers on each of 12 dedicated physical servers.

We detail the three main layers of the VLSP test-bed below.

3.3.1.1. The Lightweight Network Hypervisor (LNH) Component

The Lightweight Network Hypervisor includes a fully operational lightweight virtual router (VR) combined with virtual network connectivity. These elements can be combined in order to build any network topology required. The created virtual network is designed with the goal of transmitting and routing datagrams from any source to any destination. It behaves like a lightweight virtual network, but it has management facilities to start and stop virtual routers on-the-fly, together with the ability to create and tear down network connections

between virtual routers dynamically. Furthermore, these lightweight routers have an application layer interface that provides the capability to start and stop Java software applications. These applications use a transport protocol API which can send and receive datagrams or packets, and thus act as the service elements within the platform.

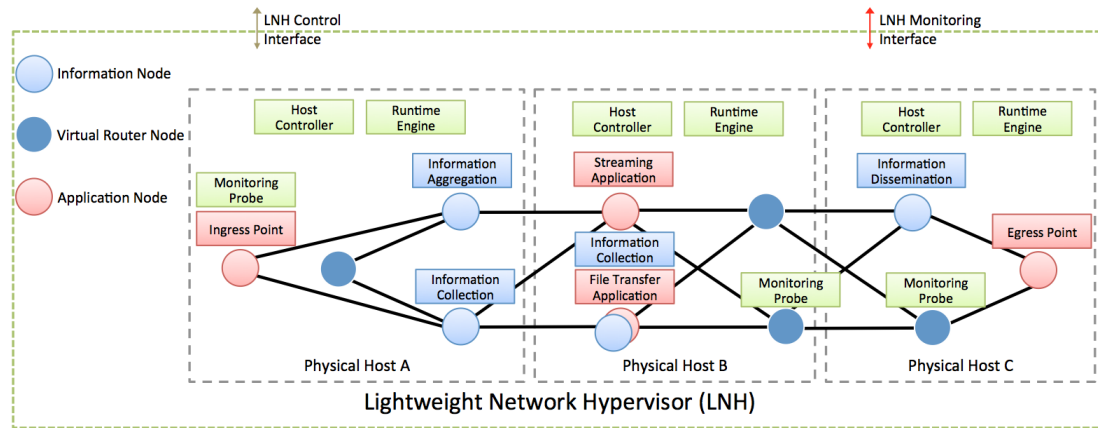


Figure 31: The Lightweight Network Hypervisor

The Host Controllers (shown in green in Figure 31) are executed on every machine that can host virtual routers. Their main job is to actually start a virtual router, stop a virtual router, start a virtual link, and stop a virtual link. Other tasks undertaken by the Host Controllers are to configure the routers once they have started, or to pass on commands to specific virtual routers, as needed.

The virtual network topology consists of virtual routers (shown as coloured circles) and virtual links (shown in black). Each virtual router is instrumented with the VLSP monitoring system in order to gather data on each of the network interfaces of each virtual router. The data includes information on traffic volumes coming in and going out of each interface. The monitoring system collects the raw data and passes it on to the Monitoring Engine function of the above layer.

The main LNH functions specification can be found in the following table.

Name	Description
Host Controllers	The host controllers execute on every physical machine and manipulate & configure virtual routers, links and virtual router applications.
Monitor Probes	The monitor probes are tiny configurable applications probing the software or hardware for monitoring data.
Runtime Engine	It is responsible for the runtime operation of the LNH, including support for event-based notifications and time scheduling.
Virtual Router Protocol Stack	The lightweight network protocol stack of the VRs.
Virtual Router Application Environment	The application environment that hosts VR applications.
Virtual Link Functionality	The functionality of the virtual links, including link

	weighting and other configuration options.
Virtual Machine for Virtual Router & Application Functionality	A virtual machine with the virtual router and the relevant applications functionality.

3.3.1.1 The Virtual Infrastructure Management (VIM) Component

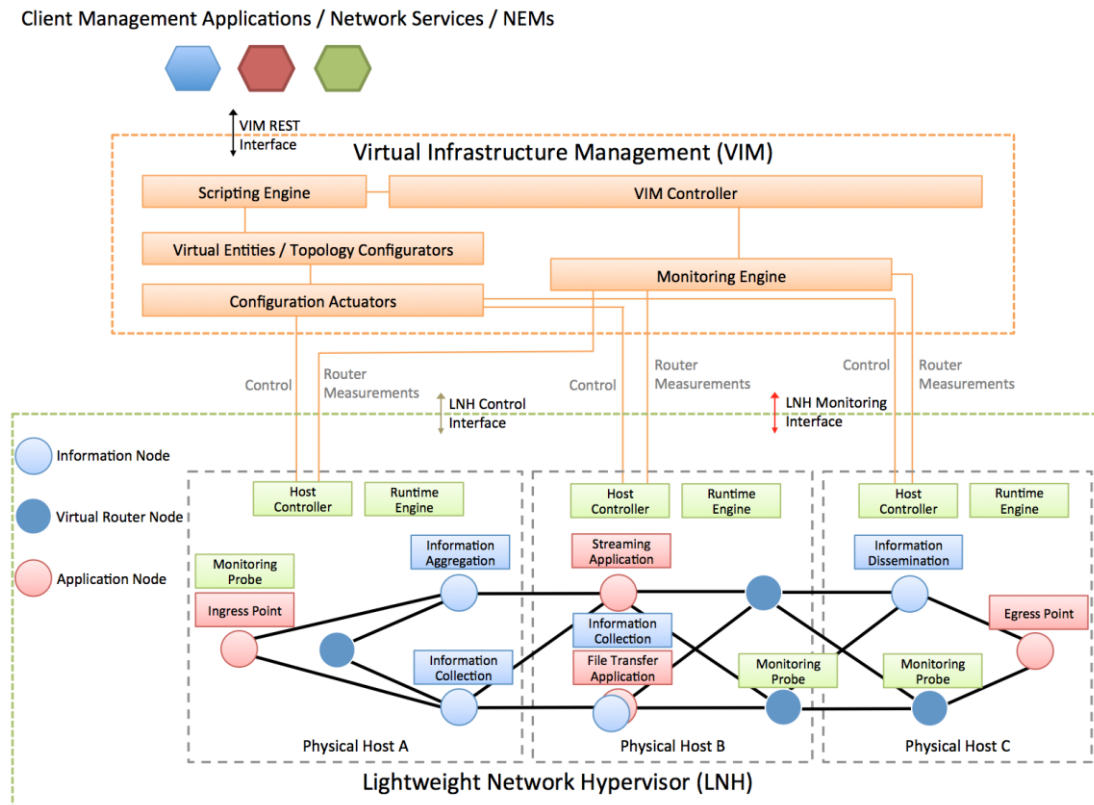


Figure 32: The Virtual Infrastructure Management Component

In this section we describe the Virtual Infrastructure Management component, highlighting its purpose and its architecture. The Virtual Infrastructure Management (VIM) component is responsible for the management and lifecycle of the virtualized elements that will be used within a network, particularly virtual network elements. As the virtual elements are not physical themselves, but exist on top of physical elements, their lifecycle and their management needs to be approached carefully to ensure continued operation and consistency. An overview of the VIM architecture on top of the LNH is shown in Figure 32.

The virtual network elements, which exist on top of physical networks, can be setup with arbitrary topologies and with an arbitrary number of end-points. The virtual links in a virtual topology are eventually mapped onto physical links in the underlying network. A virtual link may span multiple physical links, and cross many physical routers, or it may span a single physical network link. New virtual links can be added or can be removed from a virtual network dynamically at run-time.

The virtual networks are very flexible and adaptable, and generally have few limitations, except that a virtual link cannot support more traffic and higher-data rates than the underlying physical links. Furthermore, a whole virtual network can be shutdown as needed, if the applications that use it no longer need the network. Such a shutdown frees resources from the underlying physical network.

The full management of virtual networks on physical networks requires the matching and analysis of the flow rates on the virtual links to the flow rates of the underlying physical links. It is important to ensure that the physical links are not congested with too many virtual links. Also, the allocation and mapping of virtual links must take into account the current state of the physical network and the current virtual networks. However, if

a situation arises where a re-configuration is required, the virtual network management should be capable of mapping a virtual link across different physical links at run-time, but leave the virtual topology intact.

The VIM component has a seemingly simple task, but in reality the management requires continual monitoring, analysis, and adaption of the virtual elements to the physical elements. As all of these virtual elements are distributed, the management is a complex task.

The diagram in Figure 32 shows how the VIM component interacts with the virtual network elements that will be present in a running virtual network. All of the elements of the VIM component constitute a fully distributed system, whereby an element or node can reside on any host. A full virtual network can be instantiated on a single machine, for demonstration or testing purposes, or instantiated across multiple servers, in a full deployment situation.

The VIM directly controls the lifecycle of each virtual element, by collecting knowledge on the status of physical resources in order to determine where a virtual element can be created. The virtual network element will be created, managed, and shutdown by lifecycle phase of this component.

Due to the dynamic nature of virtual elements and because they can be disassociated from the physical elements they are mapped to, it is possible to do a live adaption of a virtual element from one physical host to another physical one, at run-time.

The VIM controller acts as a control point for managing the virtual elements. This block (shown in black in Figure 32) accepts all its input via the VIM REST interface from other management applications / network services or NEMs. The monitoring engine acts a collection point for the monitoring data needed to keep the management functions running. Control commands are being sent to the VIM and they are either acted upon immediately or are passed to the corresponding Host Controllers of the LNH.

The main VIM functions specification follows.

Name	Description
VIM Controller	It is the heart of the component, providing the central control of the VIM operations.
Scripting Engine	VIM can be configured using Closure scripting.
Monitoring Engine	It is the main monitoring component of the infrastructure, i.e., collecting & manipulating measurements from the monitoring probes residing at the LNH.
Virtual Entities / Topology Configurators	These functions are responsible for the configuration of virtual routers, links and topologies, supporting different levels of abstraction.
Configuration Actuators	The Virtual Entities / Topology configurators communicate with the configuration actuators which in turn enforce the configuration changes through the LNH's host controllers.

3.3.1.2 The Information & Knowledge Management (IKM) Component

The Information & Knowledge Management component is a critical part of the VLSP and UMF since it plays the role of information / knowledge collection, aggregation, storage/registry, knowledge production and distribution across all the functional components, management applications, network services and NEMs in the network environment. It can run on top of VIM, since it is fully integrated within the virtual network, e.g., the virtual routers have embedded information / knowledge manipulation capabilities. Furthermore, it is used by any client management application /network services, Network Empower Mechanisms (NEMs) or UMF core

service. As we have shown above, the IKM is fully integrated within the SDNs but acts as a standalone component as well. In the context of UMF, the IKM is the Knowledge core service and supports a wide-range of network environments beyond SDNs.

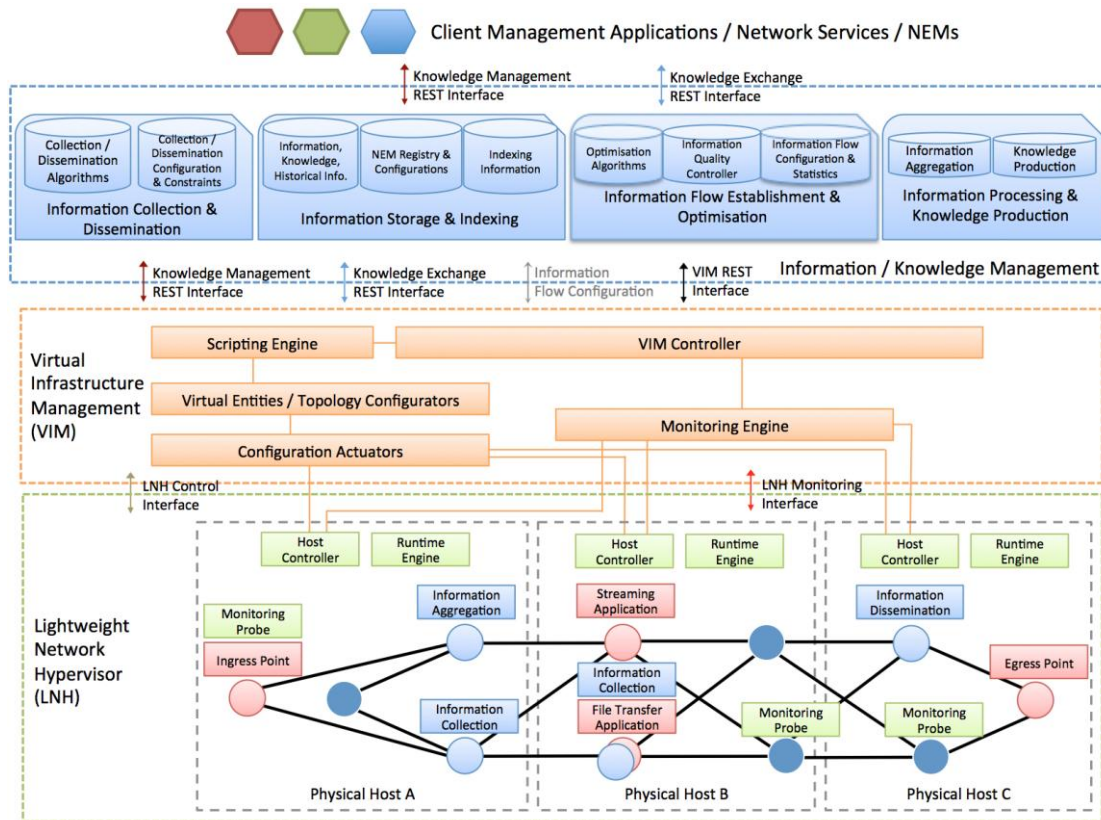


Figure 33: The Information & Knowledge Management Component

The main IKM functions can be found in the following table.

Name	Description
Information Collection & Dissemination	This function is responsible of information retrieval, sharing and dissemination.
Information Storage & Indexing	This function is a logical construct representing a distributed repository for registering information-enabled entities, indexing (and optionally storing) information/knowledge.
Information Flow Establishment & Optimization	This function regulates the information flow based on the current state and the locations of the participating entities and nodes.
Information Processing & Knowledge Production	The Information Processing and Knowledge Production function is responsible for operations related to information processing (e.g., aggregation) and knowledge production.

3.3.2 UMF management of Software Defined Networks (SDN)

The VLSP usage and integration within the rest of UMF is presented, together with a scenario showing integration with other NEMS.

3.3.2.1 The Virtual Infrastructure Management NEM Functionality

In this section we show how the Virtual Infrastructure Management NEM, is integrated with the rest of UMF, also highlighting integration with other NEMS.

In figure 6 there is a scenario where the VIM NEM manages a virtual network used for the transfer of data. There also exist two NEMS, called NEM A and NEM B, which are responsible for managing networks in completely separate domains. For example, NEM A may manage a CDN network, and NEM B may manage a wireless radio network.

The use of UMF and the integration with the 3 main management blocks (top part of figure 6) allows these NEMs to interact with each other, and also to be controlled in an organised way.

In the bottom part of Figure 34, the red line depicts some of these traffic flows across multiple networks. Within UniverSelf we have built, tested, and demonstrated such an integrated scenario using video traffic from a small CDN. This video data has flowed over a software defined virtual network that has been setup and configured using some SDN concepts. After leaving the virtual network, the video data has flowed onto a wireless network and has been displayed on various display devices.

Such integration shows the use of various NEMs in the same instantiated context, e.g. the Load Level Estimation NEM, the Core Traffic Engineering NEM, and the Virtual Infrastructure NEM.

Furthermore it highlights the integration of the data plane in the same instantiated and managed context. In the demonstration of this integrated scenario, traffic flows from a video server source in one domain, across the virtual network nodes, and into a third domain with a video player.

The Software Defined Virtual Network acts as one segment of a full network over which data will flow. Each segment of the network is managed by its own NEM. There is overall UMF management using Governance, Coordination, and Knowledge.

We have demonstrated real-time traffic flowing from one network segment, into the virtual network, and out to another network segment.

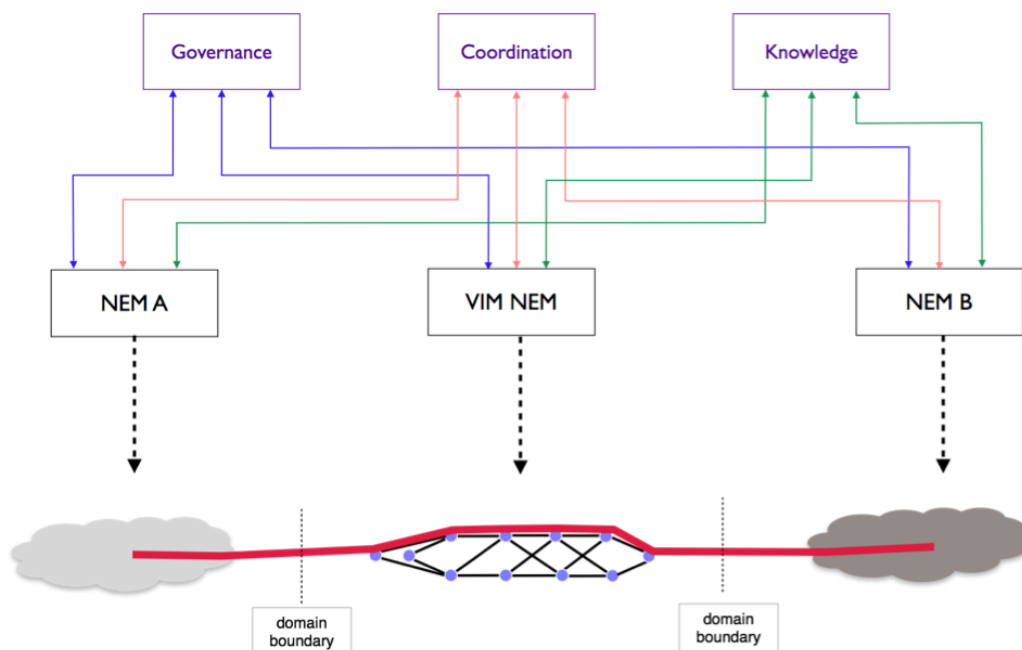


Figure 34: The VIM Integration within UMF

The combination of adaptations to various governance rules has updated the NEMs and then as a consequence the management of the network elements has been adjusted. In particular, we have used some SDN techniques to adapt the flow of the video to different display devices as the management and network environments have changed - all at on-the-fly at run-time.

The virtual network was setup incrementally: one node at a time and one link at a time. This was during the FUNEMS 2013 event, where we showed integration between many elements of the project. First, the topology of the virtual network has been created, under full management control; next an Ingress node and an Egress node were started on key nodes in the virtual network. These are special applications that are also started under management control. The Ingress and Egress applications have been specially developed to allow data plane integration between the various network segments.

The Ingress node is an application that listens on a specific port and sends any received data onto another application in the virtual network. The Egress node is an application that listens for data in the virtual network and sends that traffic onwards to a different port. A data source, such as a video server, sends its traffic to the network address of the Ingress node. The Ingress node forwards data to the Egress node, and finally the Egress node forwards the video traffic onto the video player. The full flow of traffic across the multiple domains is shown in Figure 35.

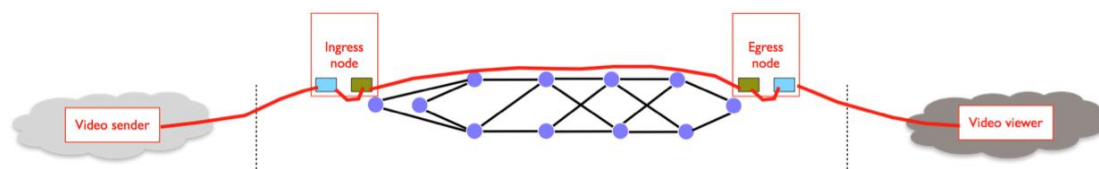


Figure 35: Traffic Flow Across Multiple Domains

The VIM NEM is not a standalone system, but acts within the scope of a full UMF system. One of the main integration aspects is with the Knowledge Block. The data and network interface measurements from the VIM NEM are passed into the Knowledge block. This data is used by other parts of UMF, such as other NEMs, for various management and reporting purposes.

The VIM NEM regularly updates the Knowledge Block with the current topology of the network. It sends separate information regarding the routers and the links. This allows any consumer of the data to build a full graph topology of a network, or just consider the routers or the links, if needed. The data is sent to the Knowledge via an information consumer and is formatted using the JSON annotation style.

For the routers the following fields are sent:

- list - which is a list of router IDs
- seq - which is a sequence number
- type - which has the value 'router' to indicate this is router information
- detail - which has a list of detail on each router. The detail for each router is as follows:
 - address - the address of a router
 - links - the IDs of the routers this router is connected to
 - mgmtPort - the port the router listens on for management commands
 - name - the name of the router
 - r2rPort - the port the router listens on for making router to router connections
 - routerID - the ID of the router
 - time - the time the router was created, in milliseconds since Jan 1 1970

An example of such data is shown below. It contains data for 5 routers.

```
{"list":[6,5,7,2,4], "seq":36, "type":"router", "detail":[
```

```

        {"address":"6","links":[2,5,4,7],"mgmtPort":11010,"name":"Router-6",
        "r2rPort":11011,"routerID":6,"time":1373635053851},
        {"address":"5","links":[4,2,6],"mgmtPort":11008,"name":"Router-5",
        "r2rPort":11009,"routerID":5,"time":1373635049618},
        {"address":"7","links":[6,4,2],"mgmtPort":11012,"name":"Router-7",
        "r2rPort":11013,"routerID":7,"time":1373635057645},
        {"address":"2","links":[4,5,6,7],"mgmtPort":11002,"name":"Router-2",
        "r2rPort":11003,"routerID":2,"time":1373635034475},
        {"address":"4","links":[2,5,6,7],"mgmtPort":11006,"name":"Router-4",
        "r2rPort":11007,"routerID":4,"time":1373635043786}}}

```

For the links the following fields are sent:

- list - which is a list of link IDs
- seq - which is a sequence number
- type - which has the value 'link' to indicate this is link information
- detail - which has a list of detail on each link. The detail for each link is as follows:
 - id - the ID of the link
 - name - the name of the link
 - nodes - the IDs of the routers this link is connected to
 - time - the time the link was created, in milliseconds since Jan 1 1970
 - weight - the link weight for the link

An example of such data is shown below. It contains data for 9 links.

```

{"list":[1048625,1572889,2293796,2097201,786457,3145777,1835044,917540,655376],      "seq":37,
"type":"link", "detail":[
  {"id":1048625,"name":"Router-2.Connection-4",
  "nodes":[2,7],"time":1373635058320,"weight":1},
  {"id":1572889,"name":"Router-4.Connection-0",
  "nodes":[4,5],"time":1373635050009,"weight":1},
  {"id":2293796,"name":"Router-5.Connection-0",
  "nodes":[5,6],"time":1373635054379,"weight":1},
  {"id":2097201,"name":"Router-4.Connection-2",
  "nodes":[4,7],"time":1373635058190,"weight":1},
  {"id":786457,"name":"Router-2.Connection-2",
  "nodes":[2,5],"time":1373635050158,"weight":1},
  {"id":3145777,"name":"Router-6.Connection-0",
  "nodes":[6,7],"time":1373635058040,"weight":1},
  {"id":1835044,"name":"Router-4.Connection-1",
  "nodes":[4,6],"time":1373635054530,"weight":1},
  {"id":917540,"name":"Router-2.Connection-3",
  "nodes":[2,6],"time":1373635054249,"weight":1},
  {"id":655376,"name":"Router-2.Connection-1",
  "nodes":[2,4],"time":1373635044318,"weight":1}}}

```

Furthermore, as the VIM NEM manages virtual routers that are instrumented and are producing live monitoring data, it is able to provide live information about the volumes of traffic on each of the interfaces of each virtual router.

For each router the following data fields are available.

- type - which has the value 'link_stats' to indicate this is link stats information
- routerID - the ID of the router
- links - the IDs of the routers this router is connected to
- seq - which is a sequence number
- link_stats - which has a list of the stats on each link. The detail for each link stat an ordered list of values. The detail is as follows:
 - name | InBytes | InPackets | InErrors | InDropped | InDataBytes | InDataPackets | OutBytes | OutPackets | OutErrors | OutDropped | OutDataBytes | OutDataPackets | InQueue | BiggestInQueue | OutQueue | BiggestOutQueue

```
{
  "type": "link_stats",
  "routerID": 2,
  "links": [33, 27, 80, 150, 122],
  "seq": 1399,
  "link_stats": [
    [
      "Router-33 Router-2.Connection-10" 189619 676 0 0 115260 565 79773 155 0 0 11016 54 0 1
    ],
    [
      "Router-27 Router-2.Connection-7" 159104 627 0 8 113220 555 140405 452 0 0 69156 339 0
    ],
    [
      "Router-80 Router-2.Connection-20" 238638 1012 0 0 193800 950 56501 69 0 0000 1 0 1,
      "Router-150 Router-2.Connection-25" 127716 548 0 0 106080 520 39043 43 0 0000 1 0 1,
      "Router-122 Router-2.Connection-24" 134112 601 0 0 118524 581 40661 45 0 0000 1 0 1]]
  ]
}
```

The VIM NEM continually produces such data, and the Knowledge Block stores such data.

3.3.2.2 The Placement Optimization NEM Functionality

The Virtual Networks are characterised as highly dynamic network environments, where topologies and nodes adapt rapidly to changes in user and service demands, user location and context changes, or resource constraints. In order to manage the challenging and dynamic infrastructures of virtual networks a monitoring system needs to be introduced which can collect and report on the behaviour of the resources; the latter should be combined with a management system that can use the monitoring information to make decisions regarding network behaviour.

The Placement Optimizer NEM (PO NEM) is responsible for optimizing the placement of special management nodes within a network. Such placement is used in various network management functions; it should be noted that the nodes, which are placed within the network can have various functions. For the NEMs described here there exist various algorithms, which take the current topology and current traffic volumes and produce as output the nodes that have been determined to be 'special'.

An example of this, consider the application placement of nodes include monitoring collection nodes and monitoring aggregations points. Such architecture uses Information Collection Points and Information Aggregation Points to scalably aggregate, filter, and collect data within the virtual domain.

The Placement Optimizer NEM takes information about the current routers and current links in a network and on a regular basis (currently configured to be 10 seconds) it applies an algorithm to determine the placement of the special nodes. It combines the router data and the link data to form a graph form of the network. It then applies one of HotSpot, Pressure, or PressureTime algorithms to determine the placement of the nodes.

As new routers and links are created by the VIM NEM, the related data is sent to the Knowledge Block. The PO NEM interacts with Knowledge, collects the latest router and link data, and updates the network graph. This interaction is seen in Figure 36.

After the network graph has been updated the PO NEM then runs the algorithm again. If there are any differences in the output from the previous run, the PO NEM injects control update commands, which are sent to the VIM NEM in order to enforce the decisions and to change the placement of the collection nodes and the aggregation point nodes.

These decisions become control commands to the VIM NEM. At 00:37 the PO NEM sets Router 2 as an aggregation point. At 00:47 it tells Router 3 to use Router 2 as an aggregation point for collected monitoring data.

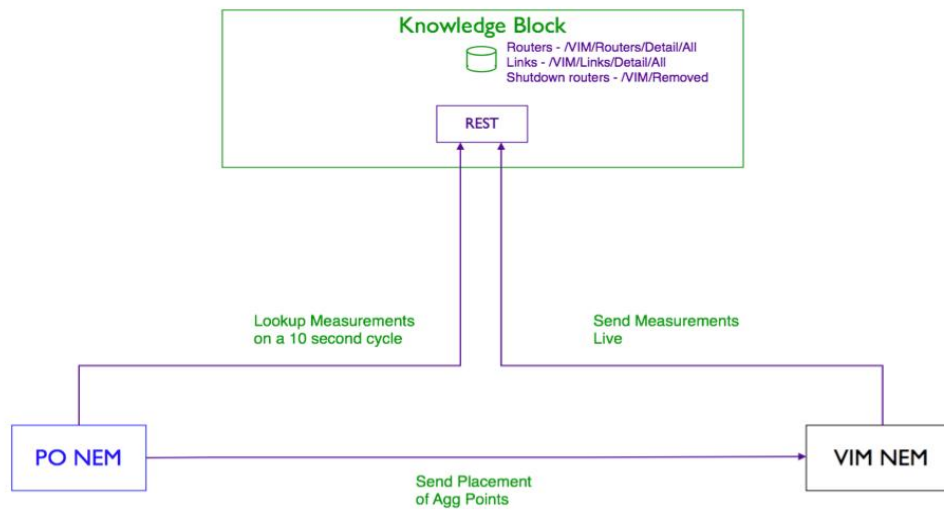


Figure 36: PO NEM Integration with VIM NEM and Knowledge Block

This work has been tested for scalability in both simulation (with over 35,000 thousand nodes) and on a custom-made virtual network testbed (with around 1200 virtual routers), where we show that it performs well at a variety of monitoring tasks. The system is scalable and that intelligent configuration of the monitoring system greatly improves its efficiency.

3.3.3 Summary

This section presents a new developed Very Lightweight Software Driven Network and Service Platform (VLSP) testbed and its use in the UMF management of Software Defined Networks (SDN).

We have shown the integration of the UMF elements into a fully working, multiple domain demonstration, showing the integration of multiple NEMs in the same context together with a Software Defined Virtual Network acting as part of a full data plane.

In the previous sections the VIM NEM was described along with its interactions with Knowledge. It was also shown how the PO NEM is able to benefit from such integration by collecting data via the Knowledge, specify its own decisions, and then update the VIM NEM. Most importantly, all of these systems are independent distributed systems, which interact via well-defined interfaces.

4 Conclusions

D4.13 presented the outcome of the deployment and implementation work within WP4, in terms of describing in detail the integrated demonstrator that combines the three use case prototypes. Such integrated demonstration has been showcased in the FUNEMS 2013 UniverSelf exhibition. This integrated demonstration aims at highlighting the real life deployment of UMF as a Network Operator tool for the efficient orchestration of a complex heterogeneous network environment.

To this end, first the deployment of UMF Core blocks has been presented, analysing the implementation of the respective functions. The latter have been complemented by analysing the implementation, instantiation and deployment details of a series of NEMs. Based on the above-described UMF core and NEM functionality, three, almost orthogonal, scenarios have been defined targeting to tackle the different project objectives. It should be noted that the UMF core blocks functionality is the same for all scenarios; however, each scenario targets at highlighting a different technical challenge of UniverSelf e.g. QoS preservation, SON, network optimization, conflict resolution etc) and therefore different NEMs have been deployed, in different and heterogeneous network domains.

D4.13 through the integrated demonstration analysis, validates the key enablers, capabilities and unified management framework developed by the project working in a live environment and real use cases reflected by the scenarios. To this end, the following project objectives have been addressed and validated through the prototype: a) the multi-facets unification, cross-technology and end-to-end view through the UMF and NEMs implementation and deployment; b) the seamless deployment and trustworthy interworking, applied on real use cases/for technology important to operators (e.g. 3GPP). This has been achieved by implementing the NEM lifecycle and NEM skin, enabling the plug-and-play installation and deployment of UniverSelf mechanisms and their dynamic interworking and communication with UMF components.

The final UniverSelf prototype will build upon the described integrated prototype. Next steps include the full-alignment with the updated version of the NEM skin, the fine-tuning of the implemented UMF functions according to the UMF final release specification and the full integration of the developed components and software modules under the scenarios. Thus, the final prototype will deliver an enhanced version of the integrated demonstrator capturing all the main project aspects and will be showcased within the Y3 review.

5 Annex A

5.1 Integrated demonstration and future steps

Towards the release of UMF as a single powerful network operator tool and the distribution of NEM and NEM SKIN as open source software, pending issues still exist. Regarding the UMF Core Blocks, their design, specification and implementation are almost completed. Integration of UMF Core blocks on a single software entity and enhancements in UMF features (GOV-PDM, H2N, COORD, KNOW, TRUST) are under elaboration. In addition, the final synthesis of UMF dashboard based on integrated demonstration data planes will be reported in D4.15

Open point regarding the distribution of NEMs is their packaging and installation in their hosting platforms. A NEM package will ideally be a single archive having a well-defined structure and containing all the necessary resources for the NEM to be instantiated (e.g. manifest, libraries) along with information regarding its dependencies (both on the software and network levels), execution peculiarities and runtime environment. A platform-independent package manager (e.g. see Maven[10]) shall be either modified to suit the UMF needs or designed and implemented from scratch.

UMF-as-a-Software (UaaS) is about providing to the UMF developers (potentially unfamiliar with the UniverSelf project) all the necessary information, tools, software and support required to become either UMF users or developers.

Although this work has been streamlined, and UMF development takes place under this mindset already, most of the effort and material needed is in progress.

6 Annex B

6.1 Instructions on building and running the Latest UMF Core Blocks and NEM prototypes

This is a short guide on building and running the latest SON Coordination prototype. For the most up-to-date version please visit[11].

Prerequisites:

- Matlab (2006 or higher has been tested) or alternatively Octave which is a GNU software that is interpreting matlab code
- JDK 1.7 or higher
- SVN client
- Eclipse (optional)

Regarding access to the SVN server[12], please refer to [13].

Building the sources

First of all you need to svn checkout the sources; 8 (Eclipse) projects independently:

- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/UMFCommon/>
- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/USelf-KnowInterface/>
- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/USelf-NEMskin/>
- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/USelf-UMF~CoreAndH2N/>
- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/Demo-Commons/>
- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/DemoCoord-Matlab/>
- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/DemoCoord-NEMs/>
- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/Demo-UMF/>

NEMs included in integrated demonstration are still in progress. An enhanced version of them may also be found in

- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/DemoGov-HandoverOptimization-NEM/>
- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/DemoGov-CoreLB-NEM/>
- <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/DemoGov-WAPMon-NEM/>

It should be noted that only NEMs' source code is available on svn, as testbed software is hardware specific and out of scope of use from future UMF users and NEM developers.

6.1.1 Setting up the simulator

As soon as you are synchronized with the sources, you need to setup your Matlab or your Octave.

Octave seems slower but it is a free software and is roughly compliant with matlab code.

Classpath.txt

In both cases your need to prepare a classpath.txt file. The one working for matlab is also working for Octave. Here is a sample [classpath.txt](#).

If you're having matlab go to Setting Matlab, you're original file is to be found where matlab is installed on your disk, precisely there: <MATLAB root dir>/toolbox/local/classpath.txt (although you are not going to make any major changes to your original classpath.txt, it is recommended to make a backup first).

If you're having Octave you have to create a file (starting from the [sample](#)) that you'll upload in your homedir.

In both cases you need to add to the original classpath.txt the last lines (right after the comment reading "Entries added for UniverSelf Demo") while making sure you replace <INSERT YOUR ROOT DIRECTORY HERE> with the root dir you checked out into. Assuming that was R:\, it should look like:

```
#
#  Entries added for UniverSelf Demo
#
R:/Demo-Commons/bin/
R:/DemoCoord-Matlab/bin/
R:/Uself-UMF~CoreAndH2N/bin/
R:/Uself-UMF~CoreAndH2N/lib/AlblfGui.jar
R:/Uself-UMF~CoreAndH2N/lib/jfreechart-1.0.14.jar
R:/Uself-UMF~CoreAndH2N/lib/piccolo2d-extras-1.3.1.jar
R:/Uself-UMF~CoreAndH2N/lib/piccolo2d-core-1.3.jar
R:/Uself-UMF~CoreAndH2N/lib/commons-codec-1.6.jar
R:/Uself-UMF~CoreAndH2N/lib/jadex_rt.jar
R:/Uself-UMF~CoreAndH2N/lib/jcommon-1.0.16.jar
R:/Uself-UMF~CoreAndH2N/lib/jcommon-1.0.16-junit.jar
R:/UMFCommon/bin/
R:/UMFCommon/lib/automaton.jar
R:/UMFCommon/lib/restbind.jar
R:/UMFCommon/lib/resty-0.3.1.jar
R:/UMFCommon/lib/simple-4.1.21.jar
R:/UMFCommon/lib/xeger-1.0-SNAPSHOT.jar
R:/UMFCommon/lib/xstream/jettison-1.3.2.jar
R:/UMFCommon/lib/xstream/xstream-1.4.3.jar
```

Also, you need to comment the line above regarding the xstream library using a sharp <#> (##\$matlabroot/java/jarext/xstream.jar in the [sample](#)).

6.1.1.1 Option 1: Setting up Matlab

Next, you need to create a shortcut to Matlab's executable that sets the working directory from the early boot. To do so, create a regular shortcut to your matlab.exe, and then set up its' properties as shown in Figure 37 (again assuming R:\ as the root directory of the projects):

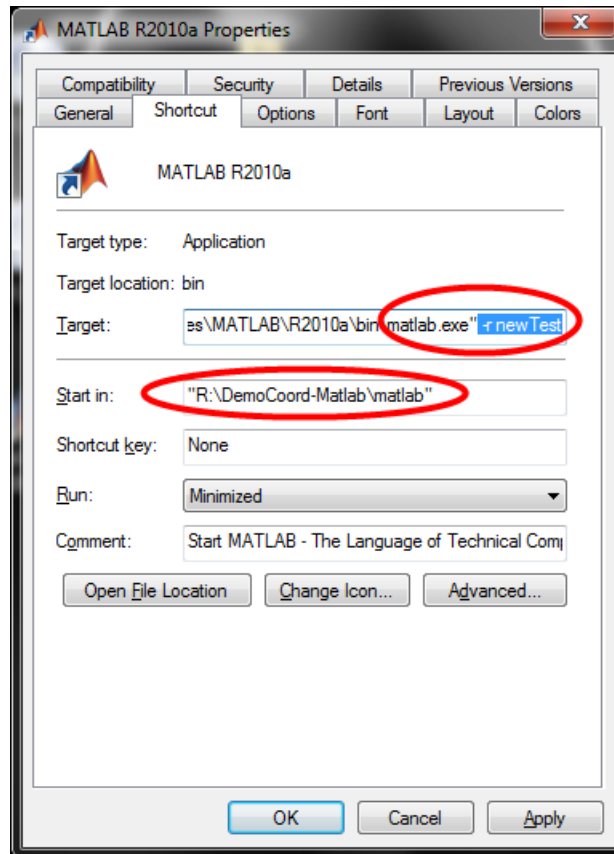


Figure 37: Configuration of the Matlab shortcut

Additionally you can make sure your matlab is using the proper version of java, for this set an environment variable in your system that will be named MATLAB_JAVA and which will contain the path to either a jre7 or a jdk7, something like: "c:\Program Files\Java\jdk1.7.0_40".

6.1.1.2 Option 2: Setting up Octave

To start with, you need installing Octave, for this please refer to the [installation page of Octave](#).

Then you need to prepare Octave for running Java, for this you can refer to [octave's wiki](#), though the 3 essential steps are provided below:

- Place the classpath.txt file in your homedir.
- Then you need to set an environment variable in your system that will be named JAVA_HOME and which will contain the path to a jdk7 (or higher - nota a jre is not enough), something like: "c:/Program Files/Java/jdk1.7.0_40", be careful that even under windows you have to use / (slashes) instead of \ (anti-slashes).
- Then you need to add to the paths of your environment variable named PATH the path of the bin dir of the same jdk, something like: "c:/Program Files/Java/jdk1.7.0_40/bin", be careful that even under windows you have to use / (slashes) instead of \ (anti-slashes).
- Then you can start Octave and in the command line type the following instruction:

```
chdir("path to matlabdir in DemoCoord-Matlab project")
```

To avoid repeating it each time Octave is started, you can place the above instruction in a .octaverc file that you will create in your homedir.

To start the simulator type in the command line:

```
newTest
```

6.1.2 Building the sources

Now it's time to build the sources. If using Eclipse, it is preferred to BuildAll (dependencies are configured automatically using subversive), otherwise, you may use buildall.bat on Windows which "calls" each project's build.bat. There are also Ant files available for packaging distributable releases (see build*.xml files and dist/ folders).

6.1.3 Running the prototype

- startMatlab from the shortcut created earlier
- runUSelf-NEMskin/umf.common.nem.weblaunch.NEMHost
- run Demo-UMF/uself.demo.DemoH2Ngui
(note that if you are running them manually, the working directory must(!) be USelf-NEMSkin and Demo-UMF for each case respectively).
- If on Linux under window manager other than Gnome, please select yours in NEMHost settings.

Unless H2N and NEMHost are running on the same machine please have a look at these [2-3 simple configuration steps](#).

Also, in the case that **H2N and Matlab are running on different machines**, H2N needs the address of the matlabrunning-machine configured into Demo-UMF/config/rest.conf, property named "uself.demo.matlabws", where "localhost" should be replaced with Matlab's IP/Hostname. Now to **start the simulation**, navigate to the "Network" tab, select "CellB", and press "Launch Simulation" (see Figure 39) from the top-left corner of the window. After a few seconds, plots should appear in the Java window of Matlab titled "Network View". Also, the "Launch Simulation" button should turn into "Kill Simulation" as soon as it's started.

6.1.4 Installation instructions of the Knowledge core service

The Knowledge core service code is fully integrated within the UniverSelf demo dashboard code, i.e., java project USelf-UMF~CoreAndH2N. So, there is no particular need for specific installation instructions besides the key-value database that supports the storage capabilities of the KNOW, i.e., we use the redis key-value database. In case redis is not running, the knowledge block runs in a limited execution mode, useful for testing purposes. The installation & execution instructions for the redis database follow.

Redis needs a linux operating system and should be compiled from its sources. It has no dependencies other than a working GCC compiler and libc. Installing it using the package manager of the Linux distribution is discouraged, as usually the available version is not the latest. The latest Redis tar ball can either be downloaded from the redis.io web site, or from the following special URL that always points to the latest stable Redis version: <http://download.redis.io/redis-stable.tar.gz>.

In order to compile Redis, the following simple steps should be followed:

```
wget http://download.redis.io/redis-stable.tar.gz
```

```
tar xvfz redis-stable.tar.gz
```

```
cd redis-stable
```

```
make
```

An optional step is to give the command *make test* to test the redis installation. After the compilation, the src directory inside the Redis distribution is populated with the different executables that are part of Redis:

- *redis-server* is the Redis Server itself.
- *redis-cli* is the command line interface utility to talk with Redis.

- *redis-benchmark* is used to check Redis performances.
- *redis-check-aof* and *redis-check-dump* are useful in the rare event of corrupted data files.

It is a good idea to copy both the Redis server than the command line interface in proper places using the following commands:

```
sudo cp redis-server /usr/local/bin/
```

```
sudo cp redis-cli /usr/local/bin/
```

(assuming that `/usr/local/bin` is in the `PATH` environment variable so both binaries can be executed without specifying the full path)

The simplest way to start the Redis server is by executing the *redis-server* binary without any argument:

```
$ redis-server
```

```
[28550] 01 Aug 19:29:28 # Warning: no config file specified, using the default config. In order to specify a config file use 'redis-server /path/to/redis.conf'
```

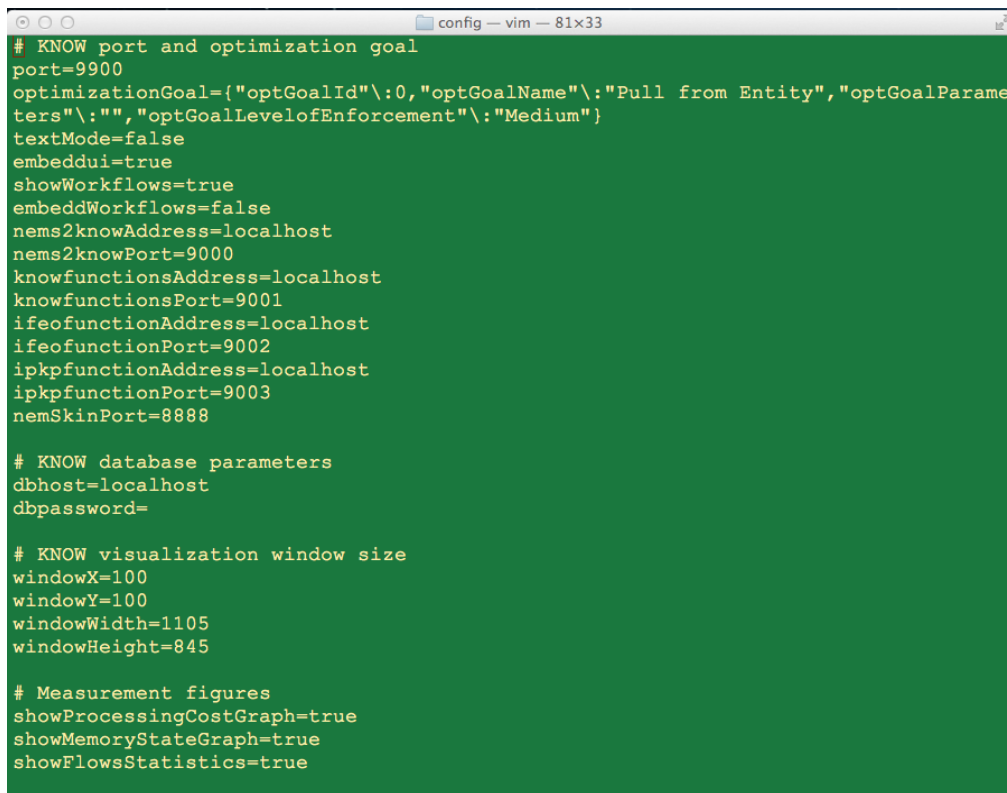
```
[28550] 01 Aug 19:29:28 * Server started, Redis version 2.2.12
```

```
[28550] 01 Aug 19:29:28 * The server is now ready to accept connections on port 6379
```

... and so forth ...

In the above example Redis was started without any explicit configuration file, so all the parameters will use the internal default ones. To start Redis with a configuration file, the full path of the configuration file should be given. The `redis.conf` file that is included in the root directory of the Redis source code distribution can be used as a template for a new configuration file. In case the access credentials for redis are changed in the `redis.conf`, the KNOW configuration file should be updated accordingly as well.

The KNOW implementation can be configured using the `know.cfg` configuration file (i.e., Figure 38). The available configuration options include: (i) KNOW GUI interface related options (e.g., whether it is embedded in the dashboard or not, visualization window size etc), (ii) the default global optimization goal used which is important for the information/knowledge flow negotiation functionality, (iii) credential information for the redis key-value database, and (iv) which real-time measurements to enable and show in the KNOW GUI.



```
# KNOW port and optimization goal
port=9900
optimizationGoal={"optGoalId"\:0,"optGoalName"\:"Pull from Entity","optGoalParameters"\:{"optGoalLevelofEnforcement"\:"Medium"}}
textMode=false
embeddUI=true
showWorkflows=true
embeddWorkflows=false
nems2knowAddress=localhost
nems2knowPort=9000
knowFunctionsAddress=localhost
knowFunctionsPort=9001
ifeofunctionAddress=localhost
ifeofunctionPort=9002
ipkpfunctionAddress=localhost
ipkpfunctionPort=9003
nemSkinPort=8888

# KNOW database parameters
dbhost=localhost
dbpassword=

# KNOW visualization window size
windowX=100
windowY=100
windowWidth=1105
windowHeight=845

# Measurement figures
showProcessingCostGraph=true
showMemoryStateGraph=true
showFlowsStatistics=true
```

Figure 38: The Knowledge Block Configuration File

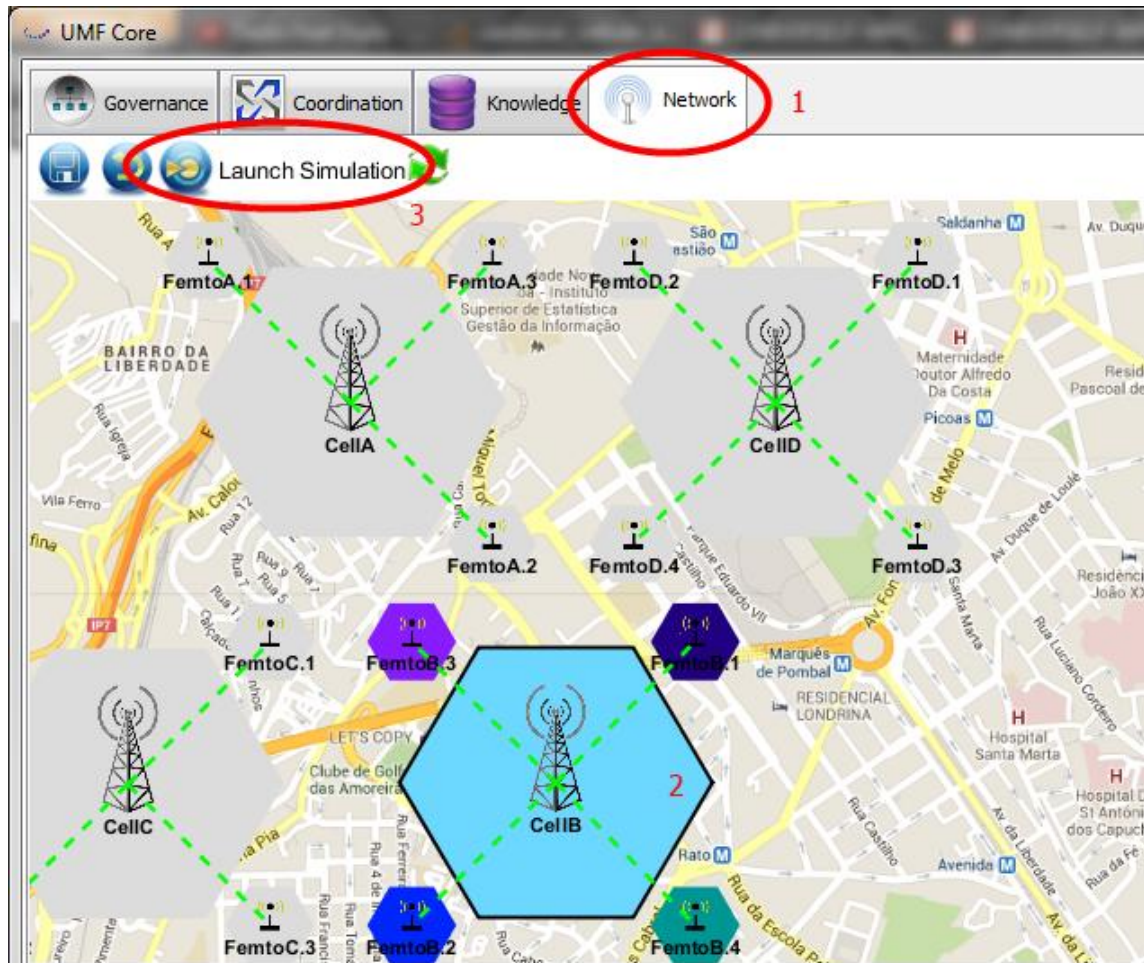


Figure 39: UMF dashboard; starting the simulation

The next thing is to **instantiate some NEMs**. Back to the "NEM Classes" list (Governance->GOV), you may select ExtendedSONBackhaulOptimNEM and SONloadBalNEM and then click on "Instantiate Selected" (please note that at the time of this writing, SONBackhaulOptimNEM is discontinued). That should bring up two new java processes and change the H2N view to the Network tab, where the NEMs should appear on the right side.

In order to **deploy these NEMs** now, you may drag'n'drop them over the simulated cell ("CellB", which in this case includes the Femtos attached to it). Note that if you attempt to deploy a NEM over a non-simulated cell (gray ones) it will crash.

In order to make it more interesting then, you may want to **unbalance the traffic** in the simulator by running DemoCoord-Matlab/uself.demo.matlab.TrafficConfigurer. The effects should be immediately visible in the simulation view.

More details on the SON functionality and the coordination mechanism can be found on paragraphs 2.3.2 and 3.2 as well as on the respective [poster](#)[14], [leaflet](#)[15] and [presentation](#)[16] given to FUNEMS 2013.

7 References

- [1] UniverSelf Deliverable 4.6, 'Synthesis of deployment results – Release 1', October 2012: <http://wiki.univerself-project.eu/deliverables/d46>
- [2] UniverSelf Deliverable 2.4, 'UMF Design – Release 3', <http://wiki.univerself-project.eu/deliverables/d24>
- [3] S. Davy, B. Jennings, J. Strassner, 'The Policy Continuum – A Formal Model', 2nd IEEE International Workshop on Modelling Autonomic Communications Environments, MACE 2007
- [4] Guerrero, A., Villagra, V.A., de Vergara, J.E.L., Sanchez-Macian, A., Berrocal, J., "Ontology based Policy Refinement Using SWRL Rules for Management Information Definitions in OWL" Proc. 17th IFIP/IEEE International Workshop on Distributed Systems, Operations and Management (DSOM), Dublin, Ireland (October 2006)
- [5] López de Vergara, A. Guerrero, V.A. Villagrà, J. Berrocal, "Ontology Based Network Management: Study Cases and Lessons Learned", Computer Science Journal of Network and Systems Management, Volume 17, Number 3, pp. 234-254, 2009
- [6] OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004. Available at: <http://www.w3.org/TR/owl-features/>. Last accessed: 15th July 2013
- [7] SWRL: A Semantic Web Rule Language Combining OWL and Rule ML. W3C Member Submission 21 May 2004. Available at: <http://www.w3.org/Submission/SWRL/>. Last accessed: 15th July 2013
- [8] UniverSelf Deliverable 4.12, 'Synthesis of deployment results – Release 2', July 2013: <http://wiki.univerself-project.eu/deliverables/d412>
- [9] Future Network and Mobile Summit (FUNEMS), Lisbon, Portugal, 2013 (<http://www.futurenetworksummit.eu/2013/>)
- [10] Apache Maven, <http://maven.apache.org/>
- [11] Latest SON Coordination Prototype installation guide, <http://wiki.univerself-project.eu/topics/5129-5246>
- [12] UniverSelf SVN Repository, <http://poseidon.ds.unipi.gr:8080/svn/repo/univerself/>
- [13] Access to the UniverSelf SVN Repository, <http://wiki.univerself-project.eu/topics/5129-5130>
- [14] FUNEMS 2013 SON Coordination Poster, http://wiki.univerself-project.eu/system/files/publicfiles/FUNEMS13-Univerself-Coord-Poster_0.pdf
- [15] FUNEMS 2013 SON Coordination Leaflet, <http://wiki.univerself-project.eu/system/files/publicfiles/FUNEMS13-Univerself-Coord-Leaflet.pdf>
- [16] FUNEMS 2013 SON Coordination Presentation, <http://wiki.univerself-project.eu/system/files/publicfiles/FUNEMS13-Univerself-Coord-Presentation.pptx>
- [17] UniverSelf Deliverable D4.4, 'First prototype of a use case'.
- [18] UniverSelf Deliverable D4.5, 'Leaflet of the first prototype of a use case'
- [19] UniverSelf Deliverable D4.8, 'Second prototype of a use case'.
- [20] UniverSelf Deliverable D4.9, 'Leaflet of the second prototype of a use case'
- [21] UniverSelf Deliverable D4.10, 'Third prototype of a use case'.
- [22] UniverSelf Deliverable D4.11, 'Leaflet of the third prototype of a use case'
- [23] UniverSelf Deliverable D4.12, 'Synthesis of deployment results – Release 2'.
- [24] UniverSelf Deliverable D2.3, 'Winning Strategies for Future Network Governances'
- [25] FUNEMS 2013, GOV scenario poster "Unified Framework for QoS and SLA-aware multi- domain self-management", http://wiki.univerself-project.eu/system/files/publicfiles/FUNEMS_2013_Univerself_QoS-aware%20multi-technology%20and%20domai%20self-management_Poster_0.pdf
- [26] FUNEMS 2013, KNOW scenario poster "Demonstration of Management of the Software Driven / Enabled Networks Using Unified Management Framework", http://wiki.univerself-project.eu/system/files/publicfiles/FUNEMS-Lisbon-MangSDN_Demo-Poster_0.pdf

8 Abbreviations

3GPP	3 rd Generation Partnership Project
API	Application Programming Interface
BHO	Backhaul Optimization
E2E	End-to-End
eNodeB	Evolved NodeB
GUI	Graphical User Interface
H2N	Human-to-Network
HetNet	Heterogeneous Network
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
KPI	Key Performance Indicator
LB	Load Balancing
LTE	Long Term Evolution
LTE-A	LTE Advanced
QoS	Quality of Service
RAN	Radio Access Network
SLA	Service Level Agreement
SON	Self Organized Networks
SVN	Subversion
UC	Use case
UaaS	UMF-as-a-Software
UMF	Unified Management Framework