Deliverable D2.2

Unified Management Framework (UMF) Specifications

Release 2

Grant Agreement	257513	
Date of Annex I	25-07-2011	
Dissemination Level	Public	
Nature	Report	
Work package	WP2 – Unified Management Framework	
Due delivery date	01 June 2012	
Actual delivery date	17 October 2012	
Lead beneficiary	UPRC	Panagiotis Demestichas, pdemest@unipi.gr

Authors	UPRC – Kostas Tsagkaris, Panagiotis Demestichas, Vera Stavroulaki, Aristi Galani, Panagiotis Vlacheas, Yiouli Kritikou, Nikos Koutsouris, Aimilia Bantouna,Dimitris Karvounas, Evagelia Tzifa, Assimina Sarli, Marios Logothetis, Andreas Georgakopoulos, Louiza Papadopoulou, Vassilis Foteinos, Dimitris Kelaidonis, George Poulios TCF – Gerard Nguengang, Mathieu Bouet ALBLF – Pierre Peloso, Samir Ghamri-Doudane, Benoit Ronot, Leila Bennacer, Magali Prunaire, Laurent Ciavaglia ALUD – Markus Gruber FT – Christian Destré, Imen Grida Ben Yahia, Zwi Altman, Richard Combes TIS – Antonio Manzalini, Roberto Minerva TID –Beatriz Fuentes Fraunhofer – Mikhail Smirnov VTT – Teemu Rautio, Jukka Mäkelä, Petteri Mannersalo, Marja Liinasuo UCL – Alex Galis, Marinos Charalambides, Lefteris Mamatas UniS –Stylianos Georgoulas, Majid Ghader NKUA - Makis Stamatelatos, Konstantinos Chatzikokolakis, Evangelos Kosmatos, Kaliroi Arapoglou, Panagiotis Spappis, George Katsikas NEC - Zarrar Yousaf

Executive summary

UniverSelf project aims at adding maturity level to the autonomic networking research field by generating high industrial impact, keeping a business focused approach and federating the various valuable research results that have already been obtained. In this context, the design of a Unified Management Framework (UMF), which targets at embedding the autonomic paradigms in any type of network in a consistent manner, shall be developed by an overall functional specification of all its components and the related underlying mechanisms.

The deliverable 2.2 presents the first complete specification of the UMF specifications. The specification focuses on: definitions of the operations and the lifecycle of the Network Empowerment Mechanisms (NEMs) that enable networks with embedded autonomic algorithms/solutions into existing and future managed networked systems and services in a "plug and play" / "unplug and play" manner; specification of the UMF core functional blocks, namely Governance, Coordination and Knowledge; and the identification of mechanisms that enable the realization of UMF functions. This specification highlights the opportunities for contributions and actions in various standardization bodies/groups, which would pave the way for industry adoption. The next release of UMF specification (deliverable 2.4) will consolidate the design artefacts and will also focus on the system architecture, deployment and migration aspects of UMF.

Table of Content

List of Figures

Foreword

Deliverable D2.2 provides a first complete functional specification of the UMF (Unified Management Framework) of the UniverSelf project, which comprises the detailed description of the Network Empowerment Mechanism (NEM) as a concept, the specification of the core UMF components and the relevant interfaces, and the possible mechanisms that can support the main functions of the core blocks.

According to the project lifecycle, the prioritized requirements prescribed in work package 4 are transferred to work package 2 to guide the specification of the Unified Management Framework (UMF). Work package 2 aims at a UMF specification in terms of identification of the required functional modules for the UMF, its interfaces and models, which also addresses the requirements deriving from the use cases handled by the project.

The UMF design is developed across three documents; each one corresponding to one UMF release, namely deliverable D2.1 (UMF release 1, published in July 2011), deliverable D2.2 (UMF release 2, published in October 2012) and deliverable D2.4 (UMF release 3, scheduled for May 2013). The scope of these deliverables, which is in line with the Description of Work and also reveals what each UMF release addresses, is as follows:

D2.1 – UMF Specifications – Release 1: The deliverable features a first description of the UMF design. It describes the foundation (requirements, objectives and approach) for achieving the target of embodying autonomic paradigms in any type of network and services, spanning widely different technological contexts, and providing to operators a service-oriented abstraction of the network they are operating. Deliverable D2.1 elaborates on the fundamental elements for achieving a network agnostic management of services, embedding advanced service and network management intelligence, and federating the management of multiple networks, hence, bridging wireless, wireline, access, core, services, etc. The fundamental elements include governance, information management, and feature embodiment (comprising the cognitive part) functions. This UMF core functions are designed with flexibility in mind to accommodate different networking scenarios and use cases in a consistent manner. It also addresses requirements deriving from the first burst of the project selected use cases. Emphasis is placed in compatibility with existing and emerging industry standards, the incorporation of recent autonomic networking research results, and in achieving a future-proof design. In particular, the UMF release 1 focuses on the identification of the common functional groups and their interfaces; the possible organization and cooperation modes between UMF elements and domains; it includes a system view of the UMF which consist of the introduction of a number of specialized logical nodes and of a possible hierarchical structure, a discussion on orchestration issues, as well as a mapping of the identified functional blocks into these nodes and the elaboration on their functionalities and interfaces among them. The positioning and mapping of the UMF (and of its components and interfaces) onto deployed and standardized control and management architectures, which is an essential aspect for the industrial impact, is initiated in this document and will be further progressed in the next releases.

D2.2 – UMF Specifications – Release 2: The deliverable is a first complete functional specification of the UMF as derived from the “bottom-up requirements” synonymous of 6 use case problem specific requirements addressing operators’ day-to-day problems identified in live networks and on existing service/network architectures; the “top-down requirements” synonymous of high-level functions, functional blocks and interfaces and “horizontal requirements” synonymous of a reposition of TMN FCAPS towards the management functions of Future Networks. A key characteristic for effective Network Empowerment Mechanisms’ deployment is based on the management framework ability to govern, orchestrate/coordinate NEMs' behaviour and facilitate the information/knowledge sharing among them. These demands lead to the need for a thorough description of the three enabling core UMF components: Governance, Knowledge and Coordination. These components incorporate key functions of the specified Functional Blocks in the first UMF release with enhancements driven by autonomic system mechanisms. In this context, UMF Specifications – Release 2 focuses on the specification of NEM definition and design, which is used then in the full description of NEM’s lifecycle, the specification of UMF core components and their interaction/interfaces, as well as, the identification of necessary mechanisms to support the main functions of the core blocks and achieve their objectives. Furthermore, the UMF information model is defined by refining and extending the TMF information framework (i.e. SID) patterns, allowing information sharing across different layers, administrative domains and network segments. The opportunities for contributions and actions in various standardization bodies/groups, which is a prerequisite for industry adoption, is presented in the deliverable 2.2 (UMF Release 2). Furthermore, the deliverable presents as an example of the UMF realization the UC6 of operator-governed, end-to-end, autonomic, joint network and service management.

D2.4 – UMF Design – Release 3: This version of the UMF will accommodate requirements from all use cases handled by the project and will incorporate corresponding network empowerment solutions for Future Networks as applicable to the overall networking infrastructure, spanning wireless and wireline, as well as access, core and service segments. Emphasis will be placed on the project-wide harmonization and consolidation of the UMF components (core components and NEMs) and on the system architecture assurances that would make UMF ready for deployment with a migration path. Deliverable D2.4 will provide the latest developments on the federation of management systems, model driven specifications, the information and knowledge management functionality and the context awareness patterns, the continuum of governance tools (cross-referencing, where appropriate, the deliverable D2.3) and the intelligence embodiment mechanisms. In addition to previous UMF releases, UMF Release 3 will focus on the complete description of the intelligence embodiment and network empowerment integration in the UMF and the network and service infrastructure; the definition of migration and deployment strategies. The document will report on the contributions to the standardization process and certification activities.

1 Introduction

The Unified Management Framework (UMF), which is developed in the UniverSelf project, is an innovative management framework that aims to solve actual network problems and address the growing management complexity of the highly decentralized and dynamic environment of resources and systems in Future Internet. The novel characteristics are achieved through the smooth and trustworthy embodiment and empowerment of autonomic principles and techniques in both services and networks.

The Network Empowerment Mechanisms (NEMs), which are introduced in the context of UMF, encapsulate autonomic functions (closed control loops/algorithms) that can be embedded into legacy and future networking systems and services in a "plug and play"/"unplug and play" way. Consequently, the UMF shall enable trustworthy integration and interworking of NEMs within the operator's management UMF ability to govern, orchestrate/coordinate different NEMs' behaviour and facilitate the information/knowledge sharing among them. These demands led to the introduction of UMF core, which consists of three enabling components, Governance (GOV), Knowledge (KNOW) and Coordination (COORD). These components incorporate key functions of the specified Functional Blocks in the first UMF release, enhanced by respective proper mechanisms. Therefore, the realization of UMF necessitates the specification of these components and their interaction/interfaces between them and with NEMs.

The main goal of this deliverable is to provide a first complete functional specification of the UMF, regarding the NEMs, the UMF core blocks and the relevant interfaces. Deliverable D2.2 shall be considered as the second release of the UMF. The prioritization dictated by the QFD analysis in Deliverable 4.2 was taken into consideration, ensuring that the respective prioritized requirements were addressed in this UMF release. Moreover, the requirements/challenges that arose from "bottom-up" (requirements derived by the set of use cases) and "top-down" (Unification & Federation, Governance, Embodiment/Network Empowerment, Service orientation, Automation/Autonomicity/Self-x and Orchestration/Coordination) methodologies of the design approach and the analysis of the state-of-the-art with respect to autonomic management/networking architectures/frameworks and "horizontal requirements" (synonymous of a reposition of TMN FCAPS towards the management functions of Future Networks), were addressed in this UMF release. This release will be complemented by the next/final release, which will be an evolved and detailed UMF specification, consolidating the developments of this release.

The document is structured as follow: Section 2 outlines the UMF functional decomposition and concisely presents the main respective components. Section 3 presents the specifications of UMF core components and NEMs, regarding their functions and their corresponding operations, as well as the relevant interfaces. Section 4 presents functional mechanisms that enable the realization of UMF core functionalities. Section 5 gives a clear view of possible opportunities for contributions and actions in various standardization bodies/groups, which is a prerequisite for industry adoption. Section 6 describes the UMF realization for the UC6 of operator-governed, end-to-end, autonomic, joint network and service management, as an illustrative example of UMF operation in practice. Section 7 presents how the requirements/challenges that arose from "bottom-up", "top-down" and "horizontal" methodologies of the design approach and the analysis of the state-of-the-art with respect to autonomic management/networking architectures/frameworks were addressed in this UMF release, along with the implied choices. Section 8 concludes the deliverable by summarising the outcomes of this second release and by elaborating on the next steps. Finally, a number of annexes provides additional information for several aspects of UMF as follows: Annex A provides a concise description of Restful UMF API; and Annex B provides definition of the data, the terms and the models that were utilized in the deliverable. References, Abbreviations and Definitions Sections are completing this document.

2 UMF overview

The rationale behind autonomics is to enable efficient and cost-effective management of networks and service infrastructures for network operators and service providers. To this end, the management and operation tasks are achieved through optimized autonomic functions, where each function is designed with a specific purpose: an operational problem to be solved, a performance objective to be achieved and a network segment or service infrastructure to be targeted. In order to highlight the role and importance of these functions, we introduce the concept of: Network Empowerment Mechanisms (NEM). A NEM encapsulates as a management application a self management function, basically a control loop or an autonomic algorithm/method. As such, the design scheme behind each NEM can be outlined as follows: use the relevant autonomic **method** to solve a concrete **operational problem** in a specific legacy **networking environment or in future networks**. **<u>NEM = method + objective + context</u>** (this definition will be further elaborated and augmented later). As examples of this triple, we can cite:

- Use of **Bayesian inference** (the method) for **fault diagnosis** (the objective) in FTTH environments (the context), or
- Use of **genetic algorithm** (the method) for **interference coordination** (the objective) in **LTE networks** (the context),
- … Further examples can be found in WP3 deliverables.

This scheme relays on the usual research approach: identify a problem within a specific context and then find/design the relevant method to address it as the basis of NEM's implementation. However, when we have to address the actual deployment of a NEM within a carrier-grade environment, further functional and non-functional requirements come into play. This introduces the main role of the UMF, which can be characterized by the following objectives: to enable a seamless integration and expandability ("plug & play" and "unplug and play") as well as to ensure a trustworthy interworking of NEMs within an operator's management ecosystem. To this end, we need:

- Lifecycle tools to deploy, drive and track activity of NEMs.
- Systemic tools to identify/avoid conflicts, and to ensure stability and performance when several NEMs are concurrently working.
- Tools to make NEMs find, formulate and share relevant information to enable or improve their functioning.

Three UMF challenging supporting functions for all NEMs are realising the above: governance, coordination and knowledge management. As a consequence, we introduce the concept of UMF core blocks in order to embody these functionalities that should be offered in a UMF ecosystem. Figure 1(a) puts in the same picture all the components at play: the three UMF core blocks (governance, coordination and knowledge), the NEMs and the network/service elements (managed elements); while Figure 1(b) presents (just as illustrative examples) the potential interactions between these components. To summarize, the NEMs are responsible for operating and managing the network and service infrastructures, while the UMF core blocks are responsible of managing and supporting the NEMs.

UMF is providing then the set of functional specifications that will make this integrated picture a reality, hence focusing on: the functional decomposition of the UMF core blocks, the requirements on the NEM structure and behaviour, the interfaces specification, as well as the workflows. The main scope of this document is to present and explain this specification work.

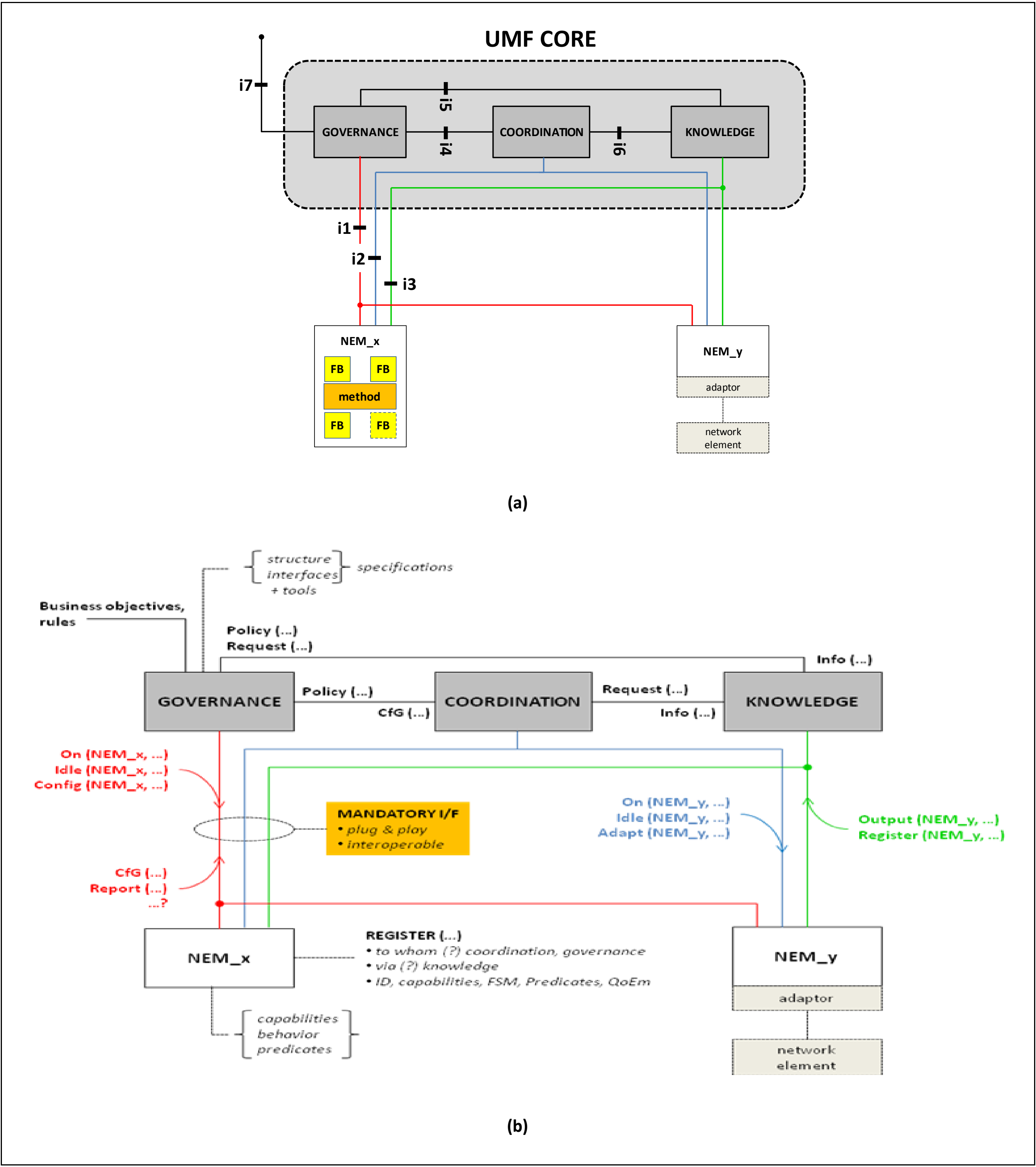


Figure 1. UMF overview and decomposition.

3 UMF functional specifications

3.1 Network Empowerment Mechanism (NEM)

First, it is important to provide a comprehensive definition of the NEM concept based on the elements and discussion presented in the previous section (UMF overview):

NEM = 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 intrinsic capability of a NEM is to be deployable and interoperable in a UMF context (e.g. an UMF compliant network).

Indeed, one of the key characteristic of UMF is to allow seamless deployment and trustworthy interworking of multiple/independent autonomic functions that will (each) ease the life of network operators. Hence NEMs can be developed by any actor of the telecommunication/networking market: equipment vendor, network management system vendor, network operator, software developers, etc. For a given NEM, the actor, who developed it, is hereafter named NEM developer.

The NEM-related specifications describe the constraints imposed by the UMF to any NEM. Hence a NEM developer will make sure the software being developed complies with these specifications in order to guarantee that the developed NEM is compliant with system instance of the UMF (i.e. deployable and interoperable in a UMF context).

In this context, and in order to understand the specification work related to NEMs, it is required to distinguish between the following concepts:

The **specifications of NEMs**, which constrain the behaviour of NEMs and define the generic part of their interfaces with UMF elements,

A **NEM class** is a piece of software that contains the logic achieving a specific autonomic function. Such class is deployed in a network running a UMF system and requires being instantiated on a set of concrete network elements to effectively perform its autonomic function,

An **instance of a given NEM class** allows performing a given autonomic function onto a given sub-set of a network. This is achieved by binding the code of a NEM class to a set of identified network resources/equipments. This NEM instance is identified by an instance ID and its unique interface with the UMF. This NEM instance at any given time is handling a set of identified network resources (this set can evolve with time). Hence there may be multiple instances of a given NEM class inside the same network e.g. one per area). A NEM instance is created by the UMF system in which it is being deployed. Moreover, a NEM instance is managed by the UMF system as an atomic entity, while its internal functioning can rely on separated piece of software running on different equipments, hence atomic NEMs are distinguishable from composite NEMs. During runtime, the distinction between these two cases is minor (limited to some more flexibility for a composite NEM regarding the flow of information), while regarding the instantiation of NEMs, the composite NEMs are stressing more importantly the process than atomic ones.

Accordingly, distinguishing between the following machine-readable descriptions of the above concepts is also required:

- A given **NEM manifest** describes a given NEM class. This description provides guidance to the network operator in order to install and configure an instance of this NEM class – the goal of a NEM manifest is similar to a datasheet). This description is issued by the NEM designer towards network operators,
- The **grammar of a NEM manifest** is a subset of UMF specifications describing which information MUST and MAY be provided by the NEM developers in order to describe their NEM class and guide its instantiation,

- A **given NEM instance description** describes a given instance of a given NEM class. This description is issued by the NEM instance towards UMF system. This description is used for registration of the NEM. It tells which information is monitored and which actions are taken.

The **grammar of a NEM instance description**, which is a subset of UMF specifications describing which information MUST and MAY be provided by the NEM instance when starting (and when its settings are changed) so as to register to the UMF system the:

- Capabilities of this NEM instance regarding information/knowledge sharing,
- Requirements of this NEM instance regarding knowledge inputs,
- Conflicts of this NEM instance with already running NEM instances of any NEM class,

A **NEM mandate** is issued by the UMF system to a NEM instance. This NEM Mandate is a set of instructions telling which equipments MUST be handled by this NEM instance and which settings this NEM instance MUST work with,

The **format of the NEM mandate** is a subset of UMF specifications describing which information MUST and MAY be provided by the UMF system to the NEM.

To illustrate the previous definitions, let's sketch a very simplified process used to start an autonomic function (coming as a NEM class) inside a UMF system. First, somehow, the software corresponding to the NEM class is being installed on the relevant machines/equipments (helped in this by the indications available in the NEM Manifest). Second, the UMF is sending to this software the mandate to create a given NEM instance, which process is completed by a NEM instance ready to register. Third, this NEM instance is sending its instance description to the UMF system in order to complete registration. Once the registration is successfully completed, the NEM instance is ready to start upon command from the UMF. This process is part of what we call **the NEM lifecycle**.

This subsection provides a detailed specification of all these concepts. First we present the lifecycle of a NEM instance with respect to UMF-compliant systems. Then, we present the information model of NEMs. Finally, we detail the different phases of the lifecycle and the different NEM state descriptions associated to them.

3.1.1 Life-cycle of a NEM instance

A NEM from the moment that it is installed until the moment that it is uninstalled is following a given life-cycle, which is specified below. Alike the life-cycle defined in OSGi for bundles, the NEM life-cycle describes the way a NEM instance can be dynamically instantiated, started, activated, halted and stopped. A simplified version of the NEM life-cycle and its different phases are presented in Figure 2.

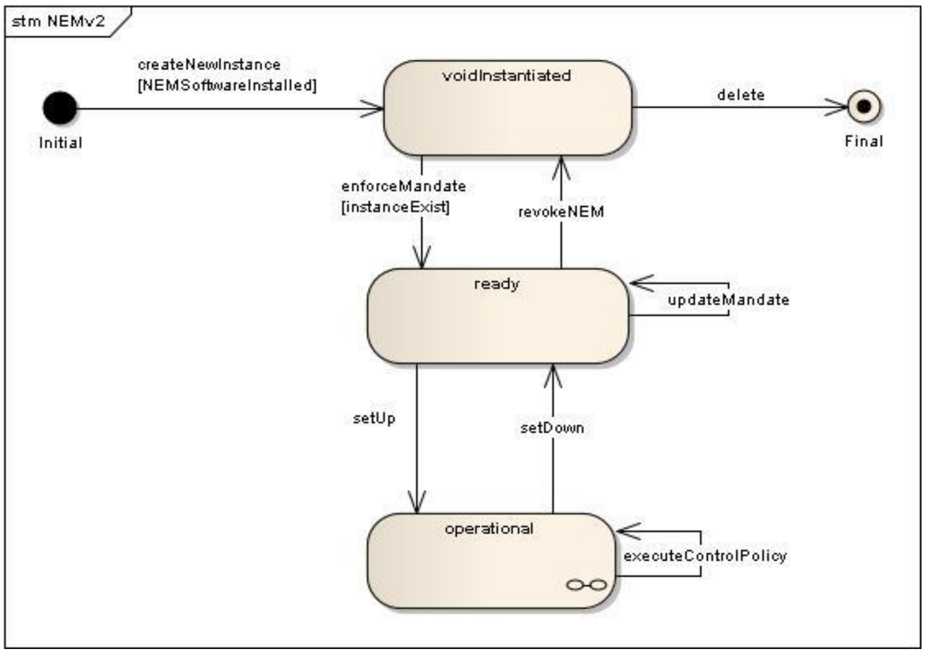


Figure 2. Simplified NEM instance life-cycle.

The NEM life-cycle consists of the following phases:

- Prior to the set-up of a NEM, when it does not exist as an instance yet, the corresponding piece(s) of software is (are) merely being installed on relevant machines, which may be used to create one or more NEM instances.
- VOID INSTANTIATED: In this first state, the NEM exits as an instance. This state is mandatory, for a NEM instance to handle a MANDATE. The MANDATE is issued by the UMF system and determines the network resources that will be managed by this instance. The MANDATE also defines the configuration options[1] applicable to this instance.
- READY: In this state the NEM instance is fully deployed but not yet operating; the appropriate pieces of software are activated on the corresponding network element and assigned to the network resources described in the MANDATE. In this state the NEM instance is also registered to the UMF core mechanisms (GOV, COORD & KNOW). All the dependencies of the NEM instance in terms of required input information (KNOW) and needed relations with other NEMs instances are identified. As a conclusion in this state, the NEM instance is known to the UMF.
- OPERATIONAL: In this state the NEM instance is operational and works under the control of COORD which is allow to set the working regime of the running instance on one of the following options:
    - achieve or not all or a part of its acquisition of information,
    - update its learning,
    - run or not its decision process,
    - share or not all or a part of its knowledge,
    - enforce or not all or a part of its actions.

The life-cycle above presents a high view of the states of a NEM. The following figure details the transitional phases, to provide a more complete NEM life-cycle.

[1] e.g. policies or constraints on behavior.

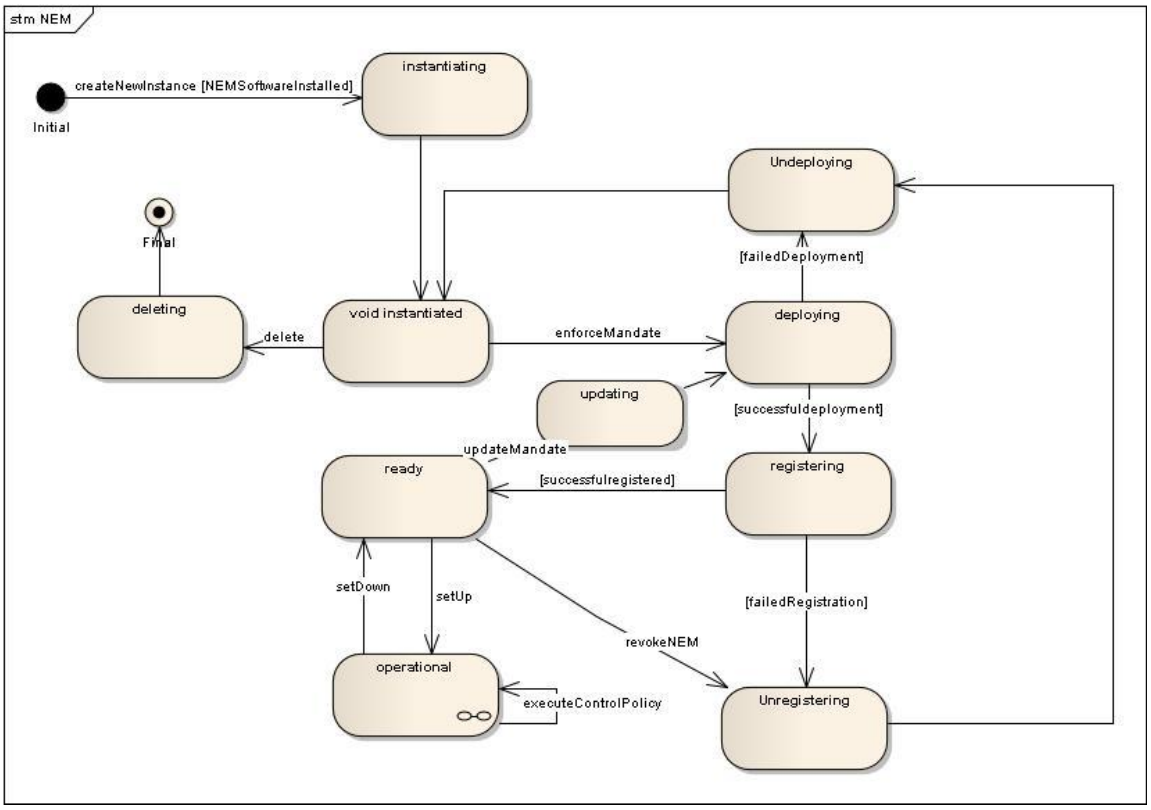


Figure 3. Detailed NEM instance life-cycle (with transitions).

When being created a NEM instance reaches a specific sub-state of INSTANTIATED that is named VOID INSTANTIATED. In this sub-state, the NEM instance is actually affected no MANDATE yet. The request named CreateNEWinstance issued by GOV to create this new instance contains a unique instance ID, which will be referred all along the NEM life. The reception of this request by the NEM instance will provide a temporary management interface for the instance. The newly created instance will listen to this interface in order to receive a MANDATE.

On reception of a MANDATE (from GOV), the NEM instance will organize itself to both handle the network resources and perform its mission (DEPLOYING trans-state). Once the deployment is completed, the NEM will achieve registration (REGISTERING trans-state), during which exchanges with GOV, COORD and KNOW will register the NEM instance. Once the registration is completed, the NEM instance is on the READY state.

On reception of a SetUp command (from GOV), the NEM instance will notify COORD of it and then move to the OPERATIONAL state.

On reception of a SetDown (from GOV), the NEM instance will abruptly stop all its processes, and then go back to the READY state.

Finally, the UPDATING trans-state is a state that is reached any time a REGISTERED[2] NEM instance receives an UPDATED MANDATE (from GOV). The NEM instance will get back to DEPLOYING.

On reception of a REVOKE (from GOV), the NEM instance will reach the VOID INSTANCE sub-state, , going through the UNREGISTERING and UNDEPLOYING states, which means all the software components involved in the NEM instance will be deactivated apart the main component. The NEM instance should be in the READY state to handle a REVOKE.

On reception of a DELETE (from GOV) the NEM instance will disappear from the UMF system. The NEM instance should be in the VOID INSTANTIATED state to handle a DELETE.

This NEM life-cycle has been designed after state of the art studies (e.g. OSGi and SOAP) and analysis of MS26 (Unification of the mechanisms embedding the UC methods) material and extended to cover the specificities

[2] actually a NEM instance, which has completed the deploying phase

related to deployment of functions over distributed systems, knowing these functions can themselves be distributed. The following sub-sections describes the initial phase of the lifecycle (NEM Manifest, NEM Installation), the NEM Instantiation to reach the VOID INSTANTIATED state, the NEM Mandate to reach the READY state and the NEM Instance Description to reach the OPERATIONAL state. Finally, the detailed operations to transit from one state to another are presented at the end.

3.1.2 Information model of NEMs

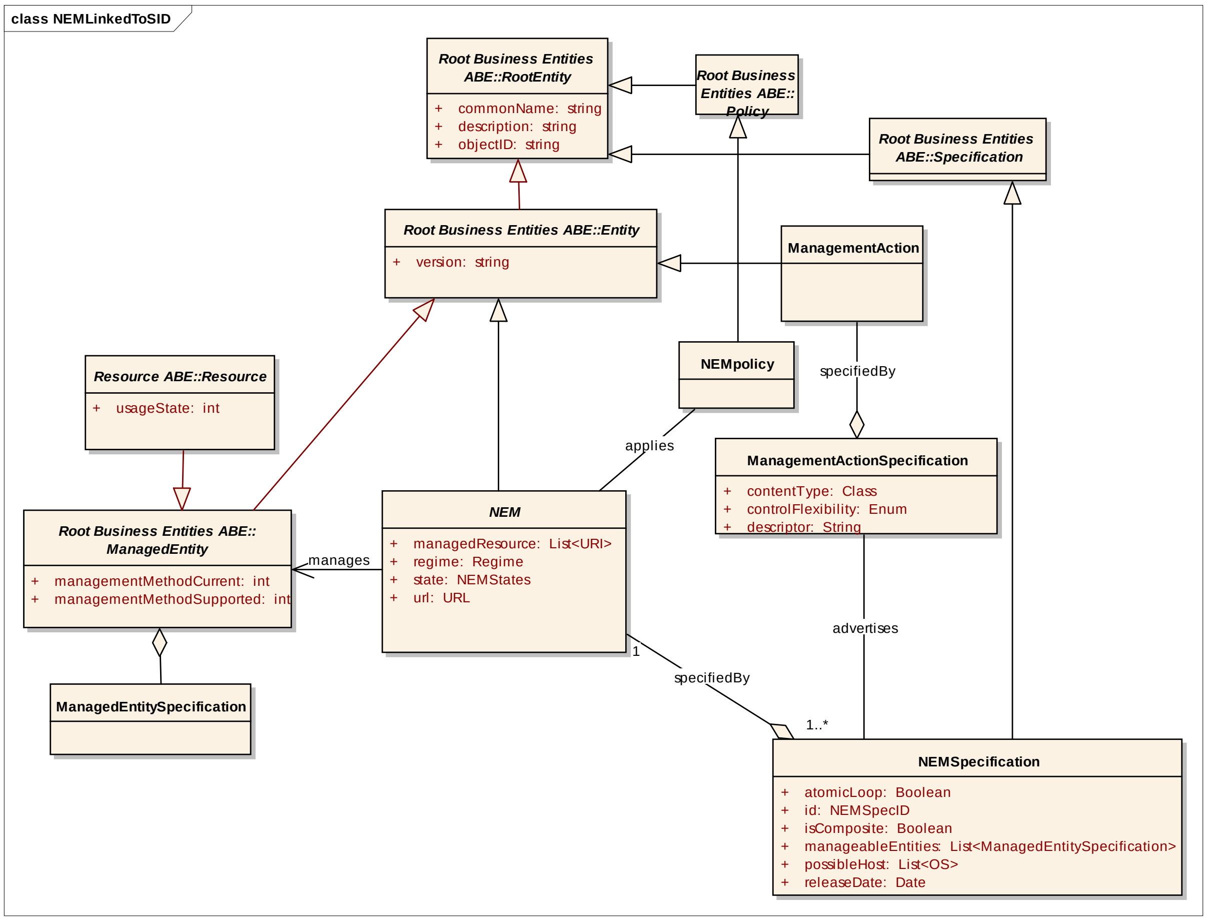

class NEMLinkedToSID
Root Business Entities ABE::RootEntity
+ commonName: string
+ description: string
+ objectID: string
Root Business Entities ABE:: Policy
Root Business Entities ABE::Specification
Root Business Entities ABE::Entity
+ version: string
ManagementAction
NEMpolicy
specifiedBy
Resource ABE::Resource
+ usageState: int
applies
ManagementActionSpecification
+ contentType: Class
+ controlFlexibility: Enum
+ descriptor: String
NEM
+ managedResource: List<URI>
+ regime: Regime
+ state: NEMStates
+ url: URL
Root Business Entities ABE:: ManagedEntity
+ managementMethodCurrent: int
+ managementMethodSupported: int
manages
advertises
1
specifiedBy
1..*
ManagedEntitySpecification
NEMSpecification
+ atomicLoop: Boolean
+ id: NEMSpecID
+ isComposite: Boolean
+ manageableEntities: List<ManagedEntitySpecification>
+ possibleHost: List<OS>
+ releaseDate: Date

Figure 4. Inheritance of UMF information model from SID (NEM part).

Figure 4 depicts the SID root diagram from which we derive the NEM concepts. The RootEntity class defines the necessary attributes that are common to define/select SID entities in the domain of service, resources as well as Policy entities. The commonName attribute enables users of the SID to refer to an object using terminology defined by their application-specific needs. The description attribute is an optional attribute that enables users of the SID to customize the description of a SID object. The objectID attribute provides a unique identity to each entity. The abstract class Entity extends the RootEntity class and represents the entities those play a business function [30].

NEM is defined as an abstract class and extends the class Entity. The “manages” association shows the link to the set of ManagedEntity managed by a given NEM.

The NEM policy is extending the SID policy class. It defines the set of policies that are applicable to a given NEM.

Following the specification pattern from the SID, NEM and NEMPolicy classes have respectively classes for NEMSpecification and NEMPolicySpecification. The specification classes describe the invariant part/information of the entity, which enables the construction of an Entity.

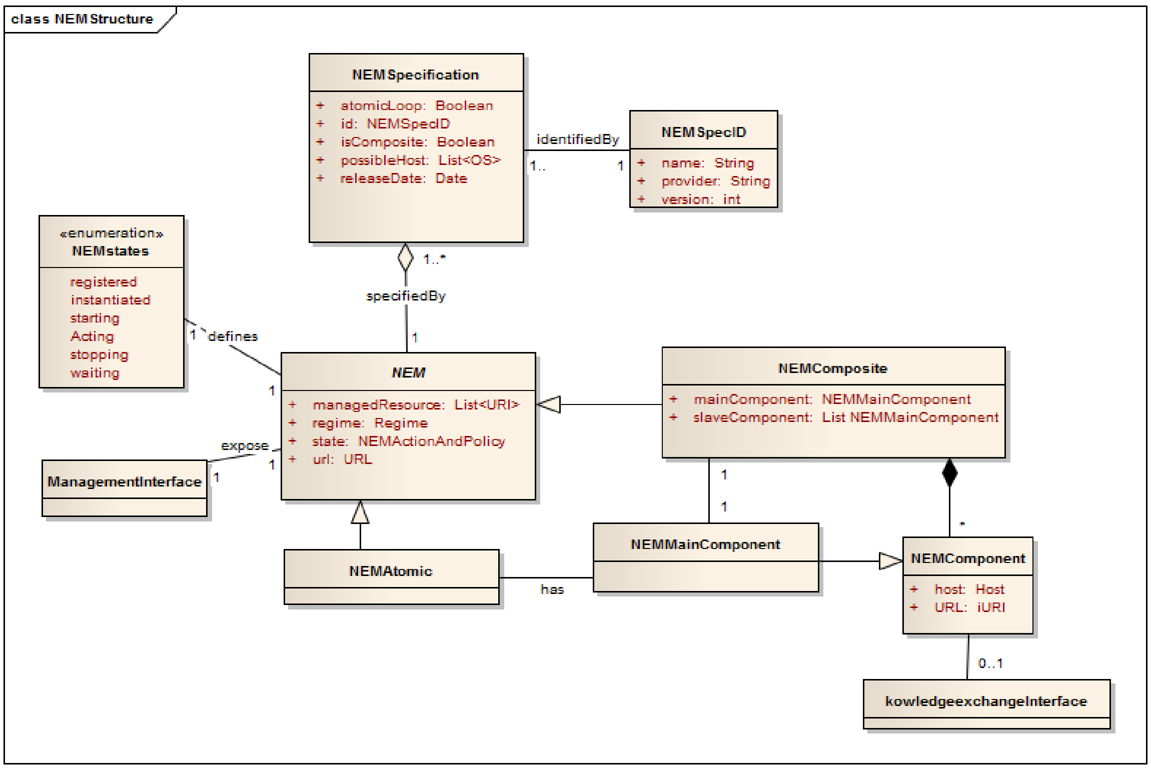


Figure 5. Representing the NEM structure in an information model view.

Figure 5 represents the structure of NEM. To start with a NEM is being specified by the attributes grouped in a NEMSpecification. Hence a NEM Manifest is merely an xml file detailing the values for all these attributes. One of the NEMSpecCharacteristics is the NEMspecID, which allows a unique identification of the "NEM class" in the catalogue as it regroups 3 attributes, which are name, provider and version. A "NEM instance" is an object of type NEM[3] exposing a management interface to be controlled by the UMF. A "NEM instance" is either atomic or composite. An atomic instance of a NEM has centralized software, and runs on a single machine, while a composite instance of a NEM has distributed software, and runs on more than one machine. This concept is slightly different from the SID pattern as the NEMComposite is not composed of multiple NEMs but of multiple NEMComponents, and a NEMAtomic is composed of a single NEMComponent. The NEMMainComponent is the one handling the control tasks of the whole NEM, meaning it is responsible for managing the relation with UMF Core Blocks and to ensure that the NEM instance as a whole is behaving accordingly to UMF instructions

A "NEM instance" is having attributes, which values are provided by either:

- The creation of the instance: Instance ID,
- The Mandate: the managedResources (the list of equipments or resources or services managed by the "NEM instance",
- Policies: the regime, etc…
- The functioning of the software of the NEM: the management interface and its URL, the NEMComponents and their KnowledgeExchangeInterfaces, which can be used to exchange information or knowledge with other UMF entities.

[3] An instanciation of the class NEM, here class refering to the class in the Information Model

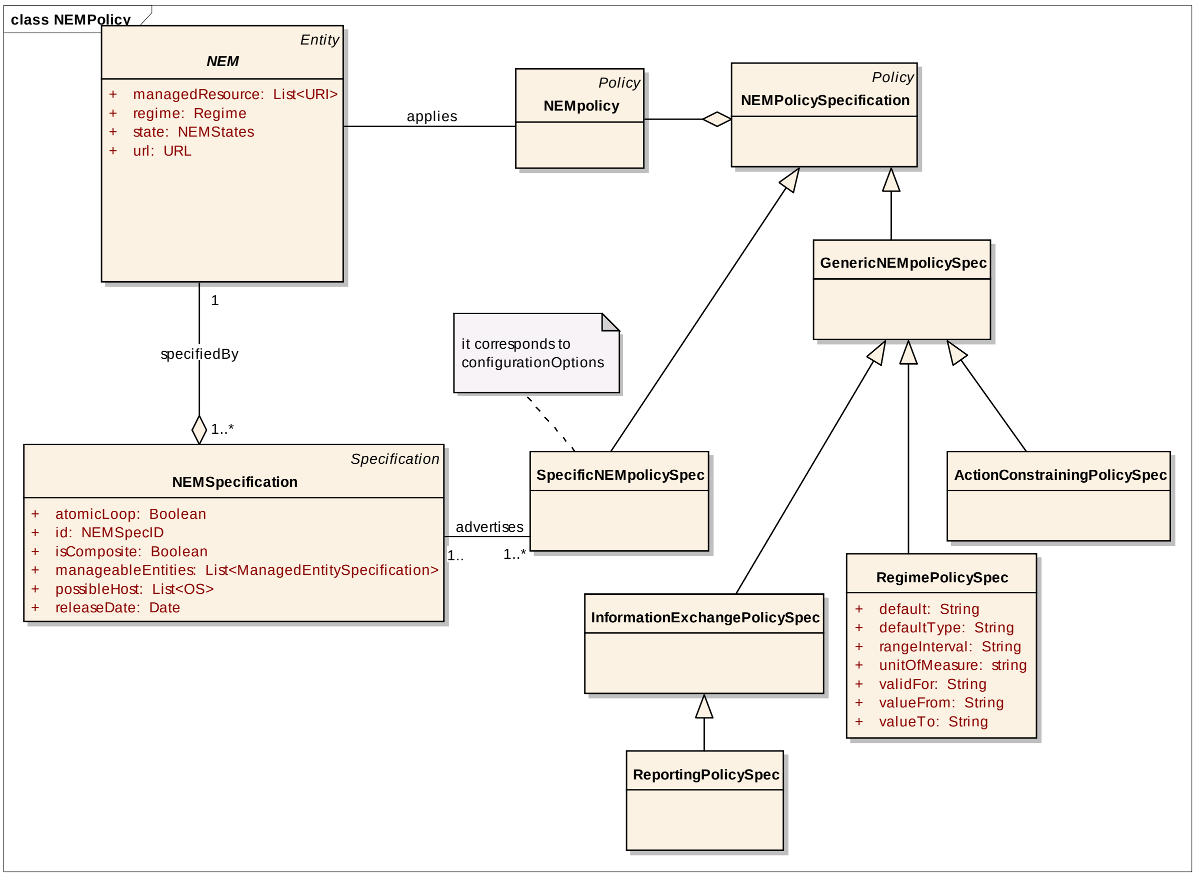


Figure 6. Information model of Policies regarding NEMs.

Figure 6 is depicting the inheritance of Policies in the scope of NEMs. Actually the picture is hiding the inheritance of policies, as it is redundant with the inheritance of PolicySpecifications (which means that for each class of PolicySpecification there is a matching class of Policy).

First of all, all the policies are inheriting from NEMPolicy.

Then there are different types of policies:

- GenericNEMPolicy is abstract, and represents all the kind of policies that are applicable to any NEM instance, for which the format is defined by the UMF specification. The exact format of these policies will be detailed in future releases of the UMF specifications.
- RegimePolicies are sent by COORD to set the regime of the NEM instance. The regime corresponds to the frequency and the modalities at which the MAPE loop of the NEM is to be run. Examples of these could be: run once every 10min, run continuously, run now only once, run when such X condition is true, etc…
- ActionConstrainingPolicies are sent by COORD to set constraints on the actions taken by a NEM instance. The goal of this can be to avoid some conflicts by providing a freedom frame to the NEM in order to avoid overlaps with conflicting NEMs. The constraints can be either to disable some specific actions, or to suspend the enforcement of the planned action to a validation by COORD or to constrain the range in which a parameter can be set. The instance description of the NEM is used to determine which subset of rules can be applied by the NEM (e.g. some NEM may provide no flexibility regarding which actions can be disabled, hence this NEM exposes itself to be simply switched in a standby mode by COORD).
- InformationExchangePolicies are sent by KNOW in order to organize an exchange of information/knowledge between UMF entities. When a NEM informs in its instance description that a given piece of information can be shared, while another NEM informs in its instance description that this same piece of information is needed to perform its analysis, then the role of KNOW is to organize

the subscription of the second NEM to the first one. The first one will not answer positively to any demand if KNOW did not previously organize this flow by setting appropriate InformationExchangePolicy (see workflows in section 3.3.4 Information Flow Establishment and Optimisation function).

- ReportingPolicies are specific InformationExchangePolicies sent by GOV to set the rules of reporting of information from the NEM instance towards GOV.
- SpecificNEMPolicies are policies, which are specific to a given NEM class. They are likely to tailor the behavior of the NEM regarding the objectives of a NEM. E.g. such a policy can be for a traffic engineering NEM a policy to set whether the objective of the traffic engineering is to save energy consumption or to avoid contention. The format of such policies is not provided by the UMF, as each NEM will have its specific. The UMF will provide a meta format, for the NEM to provide in NEMSpecificPolicySpecifications the specific format of its actual SpecificPolicies. These NEMSpecificPolicySpecifications are being advertised in the NEM Manifest.

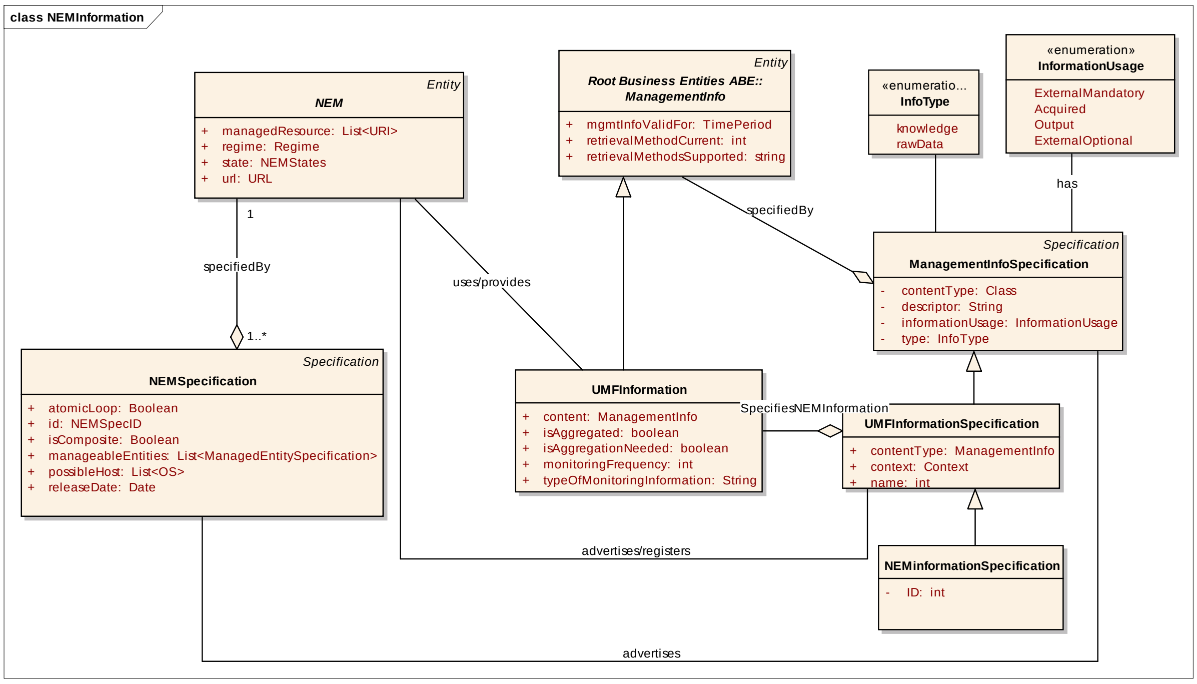


Figure 7. Information model of Information and Knowledge regarding NEMs.

Figure 7 depicts the inheritance of Information in the scope of the UMF in general and in the scope of NEMs more specifically. UMFInformation objects are exchanged between UMF through one of the Knowledge Exchange workflow (see workflows in section 3.3.4 Information Flow Establishment and Optimisation function). A NEM can be at one or the two endpoints of such an exchange.

Figure 7 depicts three levels regarding information:

1. ManagementInformationSpecification: This level depicts the nature of the information, e.g. "Load of link (in Bit/s)". This class of the information model is used to build catalogues of information E.g. The list of the nature of all the information acquired by a given class of NEM, which corresponds to the Acquired_Inputs field of the NEM Manifest (see section 3.1.3), similarly for the following fields of the Manifest: Optional_External_Input, Mandatory_External_Input and Available Outputs.
   A NEM agnostic catalogue should be built to fill an ontology describing the relations between the different entities of the network. This ontology could describe that "load of link (in %)" is related to "link capacity" which is the "sum" of "ports capacity" "composing" the "link". This ontology would be used to help COORD identify conflicts between NEMs. The ontology should stay at the level of the ManagementInformationSpec.

2. UMFInformationSpecification: This level designates exactly the information, e.g. "The load of the link between router 1.1.1.1 and router 2.2.2.2". This class of the information model is used to build catalogues such as:
   - the indexation in KNOW of all the available outputs of every NEMs (used to perform the identification of the providing entity when organizing knowledge exchange with other UMF entities – see workflows in section 3.3.4 Information Flow Establishment and Optimisation function),
   - the indexation in COORD of inputs of NEMs to identify conflicts with other NEMs,
   - Instance Description disclosed by NEM instances when registering (which are then indexed by COORD and KNOW – see needs above), namely the Available_Outputs, Optional_External_Input, Mandatory_External_Input and Acquired_Inputs fields (see section 3.1.6).

   UMFInformationSpecification are extending the ManagementInfoSpecification with the context attribute (in the above example the designation of the link: router 1.1.1.1 to 2.2.2.2). The context class is taken from DEN-ng extensions disclosed in the following paper [1].
3. UMFInformation: This class represents the information actually exchanged through a Knowledge Exchange Interface (see workflows in section 3.3.4 Information Flow Establishment and Optimisation function). For this exchange to happen KNOW takes in charge its organization, which will be materialized by an Information Policy (see Figure 6).
   This is a class inheriting from ManagementInformation (defined in SID) that is being specified by an UMFInformationSpecification. This is then a ManagementInformation enriched with a context (in order to know that the load which is 70% is actually referring to the link between router 1.1.1.1 and router 2.2.2.2.). The actual value is of any sub-class of ManagementInformation as defined in SID. The ManagementInformationSpecification is actually describing with its attribute contentType which sub-class of ManagementInformation will be used to describe the value of the UMFInformation.

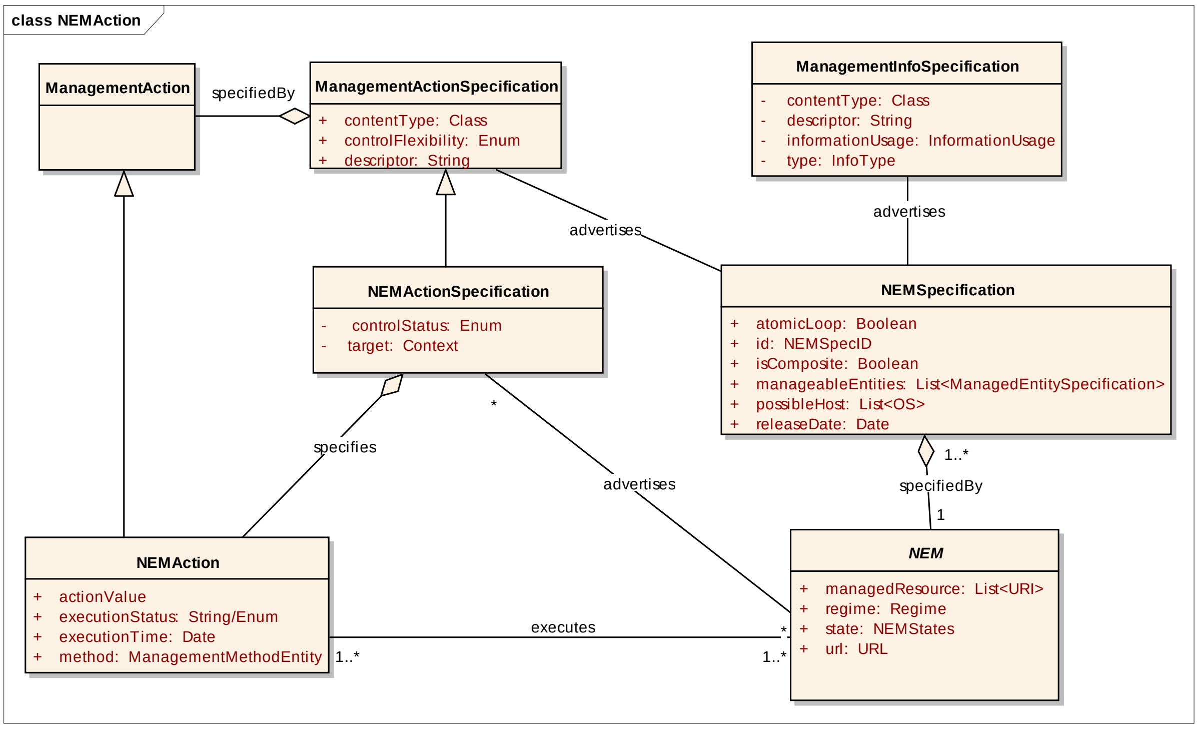


Figure 8. Information model of Actions regarding NEMs.

Figure 8 depicts the inheritance of Actions in the scope of the UMF in general and in the scope of NEMs more specifically. NEMActions are executed by NEMs onto ManagedEntities (resources or services). These correspond to the change in settings of the services or equipments that NEMs are performing.

Specifically, it depicts three levels regarding the actions:

1. ManagementActionSpecification: This level depicts the nature of the action, e.g. "Switch on/off a port". This class of the information model is used to build catalogues of actions e.g. the list of the nature of all the actions potentially performed by a given class of NEM, which corresponds to the Possible_Actions field of the NEM Manifest (see section 3.1.3).
   A NEM agnostic catalogue should be also used to complete the ontology describing the relations between the different entities of the network. This ontology could describe that "switching on/off a port" is changing "link capacity" if "port" is "composing" the "link".
2. NEMActionSpecification: This level designates exactly the action, e.g. "Switch on/off the port 12 of router 1.1.1.1". This class of the information model is used to build catalogues such as:
   - the indexation in COORD of actions of NEMs to identify conflicts with other NEMs,
   - Instance Description disclosed by NEM instances when registering (which are then indexed by COORD and KNOW – see needs above), namely the Possible_Actions field (see section 3.1.6).

   NEMActionSpecification are extending the ManagementActionSpecification with the context attribute (in the above example the designation of the port 12 of the router 1.1.1.1). Alike the UMFInformationSpecification, the context class is taken from DEN-ng extensions.
3. NEMAction: This class represents the action actually performed by the NEM. It then contains the value of the action, which in our above example can be either On or Off. The NEMActionSpecification describes (with its controlStatus attribute) which is the allowed control of this action, while the ManagementActionSpecification describes (with its controlFlexibility attribute) which are the allowed control of this kind of action (this property only depends on the flexibility offered by the NEM designer at implementation time). The usage of these control level are explained in section 3.1.8 NEM's Relations with Coordination.

3.1.3 NEM Manifest

A NEM class is being described by its Manifest, which is machine readable. This Manifest provides information (such as the type of network equipments that can be handled, the identification of the NEM class) for the operator to deploy the NEM in its infrastructure. This Manifest could be used:

- as soon as a NEM is purchased, as it contains most of the technical details of the NEM,
- when organizing the network management in order to determine the NEM deployment map,
- at deployment time, in order to generate the Mandate that will be sent to the NEM instance,
- any time during the life of a NEM instance.

Table 1. Format of NEM Manifest

Field Name	Type	Description
ID	NEM Spec ID	To have a unique identifier of the NEM class
Name	String	Name of the NEM class
Provider ID	String	Name of the NEM developer (name of the company)
Version	Int[]	Version of the NEM
Release Date	Date	Date of release of the NEM
Features	String	Text field used to describe what is the feature achieved by the NEM
User Guide URL	URL	Optional - Used to have a link onto a web server providing guidance for the use of the NEM
Possible Hosts	List<OS>	Lists the OS on which the NEM (or more precisely the NEM Component) can be installed
Manageable Entities	List<Managed EntitySpecification>	Lists the type of equipments/services that can be managed by the NEM
Is Composite	Boolean	Depicts whether the NEM is atomic or composite
Is Atomic Loop	Boolean	Depicts whether the algorithm of the NEM works as a single

		control loop or as a set of cooperating control loops. (This information makes sense in order to achieve joint optimization, then the NEM delegates its utility function to a UMF mechanism, in case a NEM is set to false there, then it will delegate a set of local utility functions).
Acquired Inputs	List<Management InfoSpecification>	Lists the nature of information acquired by the NEM itself
Optional External Inputs	List<Management InfoSpecification>	Lists the nature of information that the NEM should receive from KNOWLEDGE (directly or indirectly)
Mandatory External Inputs	List<Management InfoSpecification>	Lists the nature of information that the NEM must receive from KNOWLEDGE (directly or indirectly)
Available Outputs	List<Management InfoSpecification>	Lists the nature of information that can be provided by the NEM to any UMF entity. This list does not repeat what can be deduced from the other fields of the manifest, i.e. every acquired input can be shared.
Possible Actions	List<Management ActionSpecification>	Lists the nature of actions that the NEM can apply onto the managed entities
Configuration Options	List<Specific NEMPolicySpec>	Lists the configuration options that can be applied to the NEM. The NEM specific policies must be depicted here.

Hereafter is an indicative example of the information which comprises a NEM Manifest, namely for the Green TE NEM.

```
<eu.univerself.nem.Manifest>
    <NEMspecID>
        <Name>Green TE</Name>
        <Provider>StylianosCorp</Provider>
        <Version>1.0.0</Version>
    </NEMspecID>
    <Features>This NEM is achieving a Traffic Engineering function that is saving
       energy consumption of an IP network. It selects links and ports to be put
       into sleep based on traffic demand and link utilization/connectivity
       constraints.</Features>
    <releaseDate>2012-07-23 11:25:32.647 UTC</releaseDate>
    <UserGuideURL>www.stylianoscorp.com/support/GreenTE</UserGuideURL>
    <isAtomicLoop>true</isAtomicLoop>
    <isComposite>false</isComposite>
    <PossibleHosts>
        <OS>UnixOS</OS>
    </PossibleHosts>
    <ManageableEntities>
        <ManagedEntitySpecification>ALU SAR7705</ManagedEntitySpecification>
        <ManagedEntitySpecification>ALU 7710</ManagedEntitySpecification>
        <ManagedEntitySpecification>ALU SR7750</ManagedEntitySpecification>
        <ManagedEntitySpecification>Cisco CRS-1</ManagedEntitySpecification>
        <ManagedEntitySpecification>Cisco CRS-2</ManagedEntitySpecification>
       <!—Relatively to the tag <ManagedEntitySpecification> to be accurate there,
       this XML file is providing an id field of a ManagedEntitySpecification, this
       id field allowing to pick the proper managedentityspecification from the
       corresponding catalogue -->
    </ManageableEntities>
    <AcquiredInputs>
       <!—Relatively to the tag <ManagementInfoSpecification> for sake of
       readibility of the example, it is just a lightweight version that has been
       provided here, the full format contains attributes, which  are being
       described in the information model, namely : descriptor, contentType,
       informationUsage and type -->
        <ManagementInfoSpecification
           contentType="EthernetPortInfoSpecification">Description of router
           port(ID, capacity)</ManagementInfoSpecification>
        <ManagementInfoSpecification
           contentType="IPInterfaceInfoSpecification">Description of router
           interface (ID, capacity, List<Ports ID>, IP@)
           </ManagementInfoSpecification>
        <ManagementInfoSpecification contentType="Numeric">Load of router interface
           </ManagementInfoSpecification>
        <ManagementInfoSpecification contentType="List<LSA>">Routing Table
           </ManagementInfoSpecification>
    </AcquiredInputs>
    <OptionalExternalInputs>
        <ManagementInfoSpecification contentType="Numeric">Prediction of router
          interface load </ManagementInfoSpecification>
    </OptionalExternalInputs>
    <PossibleActions>
       <!—Relatively to the tag <ManagementActionSpecification> for sake of
       readibility of the example, it is just a lightweight version that has been
       provided here, the full format contains attributes, which  are being
       described in the information model, namely : descriptor, contentType,
       controlFelxibility -->
        <ManagementActionSpecification contentType="Boolean">Switch ON/OFF Ethernet
          port </ManagementActionSpecification>
        <ManagementActionSpecification contentType="Boolean">Switch ON/OFF IP
          interface </ManagementActionSpecification>
        <ManagementActionSpecification contentType="Numeric">Change metric of IP
          interface </ManagementActionSpecification>
    </PossibleActions>
    <ConfigurationOptions>
        <SpecificNEMPolicy>
            <name>GreenTimelyThreshold</name>
            <description>Minimal time under which no-switchoff will occur
              </description>
            <defaultValue>15</defaultValue>
        </SpecificNEMPolicy>
       <!--This is just an example, as the internal format of these policies is not
       specified yet-->
    </ConfigurationOptions>
</eu.univerself.nem.Manifest>
```

3.1.4 NEM Installation and Instantiation

The initial phase consists of installing the piece of code of a NEM onto the relevant hosts. At least 3 different scenarios can be considered for that:

1. The code of the NEM is embedded inside the controller of a given type of network equipments/resources,
2. The code of the NEM is manually[4] copied by a network operator into hosts inside the network. The hosts can be servers or network equipments allowing uploads,
3. The code of the NEM is copied into a specific GOV repository, from where it will be autonomously copied to the relevant hosts.

The UMF release 2 is not specifying any of these installation scenarios, but the creation of a new NEM instance is specified hereafter. Once being installed on the hosts, a kind of "code loader" will take part in the creation of the instance as its role is to handle a CREATE NEW INSTANCE command from GOV and to load the required components of NEM. For this purpose:

- A NEM MUST be provided with its code loader.
- A code loader SHOULD be capable of creating more than one instance of a given NEM class.
- A code loader MAY have the capability to load more than one class of NEMs (as long as GOV associates the code loader to each of these NEMs).
- There MAY BE more than one code loader for a given NEM class.
    1. GOV MAY know more than one loader,
    2. Each loader MUST have the intrinsic capability to communicate with other loaders of the same NEM class,
    3. Each loader SHOULD be capable to communicate with any loader of this NEM class activated in the system covered by the same UMF, restrictions may come from:
        - The structure of the communication infrastructure may block this communication,
        - Lack of awareness of other loaders (installation of the loader does not impose an exhaustive knowledge of any other loaders of the same class, though this is preferred.

[4] Manually, may mean either physically or remotely.

- GOV MUST know (the interface of) at least one code loader of this NEM class in order to create a NEM instance of a given NEM class.
- When receiving the NEW INSTANCE command, the code loader MUST create a VOID INSTANCE, which means:
    1. It MUST at least provide an answer to GOV indicating an interface on which GOV CAN send the NEM MANDATE,
    2. This interface MUST BE capable of handling a NEM MANDATE of this NEM class and MUST respond negatively to a NEM MANDATE of a different NEM class.

A CREATE NEW INSTANCE message is actually a specific case of a NEM INSTANTIATION/DELETION message that follows the format described below:

Table 2. Format of NEM INSTANTIATION/ DELETION message

Field Name	*Type*	*Description*
Class ID	NEM Spec ID	The identification of the NEM class
Instance ID	Integer	The unique ID provided by the UMF to identify this NEM instance.
Action	ENUM	This field is used to communicate the action that can be either: NEW INSTANCE or DELETE INSTANCE.

Then the "NEM loader" is responding with a message following the format below:

Table 3. Format of NEM INSTANTIATION/DELETION response message

Field Name	*Type*	*Description*
Instance ID	Integer	The unique ID provided by the UMF to identify this NEM instance
Result	ENUM	States whether the action was successful or not
Management @	URI	The address of the NEM Management interface, this field is optional, as it contains content only when the response is successfully answering to a NEW INSTANCE action

3.1.5 NEM Mandate

A **NEM mandate** is issued by the UMF system to a NEM instance. This NEM Mandate is a set of instructions telling which network equipments MUST be handled by this NEM instance and which settings this NEM instance MUST work with.

Table 4. Format of NEM Mandate

Field Name	*Type*	*Description*
GOV@	URI	To exchange with GOV UMF Block
COORD@	URI	To exchange with COORD UMF Block
KNOW@	URI	To exchange with KNOW UMF Block
Managed Entities	List<URI>	Listing all the equipments/services that the NEM has to handle, monitor, optimize, etc... after being successfully deployed.
Configuration Options	List<Policy>	Listing chosen values for generic or specific options

Hereafter an indicative example of the information comprised in a NEM Mandate, namely for the creation of a Green TE NEM instance.

```
<eu.univerself.nem.Mandate>
    <Instance ID>356789456</Instance ID>
    <GOV_address>1.1.1.1</GOV_address>
    <COORD address>2.2.2.2</COORD address>
    <KNOW address>3.3.3.3</KNOW address>
    <Instance ID>356789456</Instance ID>
    <ManagedEquipments>
        {127.100.50.1 , 127.100.50.5 , 127.100.50.15 , 127.100.50.19 ,
      127.100.50.36}
        <!--The 3 first happen to be ALU SR7750 and the 2 last happen to be Cisco
             CSR-1.-->
        <!--This is a lightweight format to provide a list of URIs, this could
             alternatively be expressed as
            <URI>127.100.50.1 </URI> etc...
        -->
    </ManagedEquipments>
    <Configuration Options>
        <SpecificNEMPolicy>
            <name>GreenTimelyThreshold</name>
            <value>10</value>
        </SpecificNEMPolicy>
        <ReportingPolicy>
            <ReportInterval>30</ReportInterval>
        </ReportingPolicy>
        <!--These are just examples, as the internal format of these policies are
          not specified yet-->
    </Configuration Options>
</eu.univerself.nem.Mandate>
```

The deployment of a NEM instance MUST happen accordingly to the MANDATE. When receiving the MANDATE the NEM instance is not even deployed. There may be more than one possible host for the code of the NEM, there may be multiple ones working together.

Following what is depicted in the life-cycle of a NEM (see section 3.1.1), a NEM Mandate can be sent to a NEM instance, when:

- The NEM Instance is void instantiated; then the MANDATE is enforced as being a completely new one.
- The NEM instance is in the Ready state; then the previous MANDATE is updated with the content of this mandate. As this may imply redeploying and reregistering of the NEM, this operation cannot happen while a NEM may be actually working under the control of COORD, which prevents the update of a MANDATE in the Operational state.

The MANDATE determines a list of network equipments. The installation phase had already determined a set of hosts capable of running a software component of the NEM class. The deployment of a given NEM instance corresponds to:

- finding suitable hosts (machines to run the software component on and where the code loader can start the code),
- activating in these hosts the software component(s), (role of the code loader)
- associating to those a sub-set of the equipments,
- additionally, federating these software components into a unique entity by the selection of a leader,

This process may change the interface of the NEM, as it MUST be the interface of the leading software component. This new interface will be advertised through registration inside the instance description.

3.1.6 NEM Instance Description

A **given NEM instance description** describes a given instance of a given NEM class. This description is issued by the NEM instance towards the UMF system. This description is used for registration of the NEM. It tells which information is monitored and actions are taken by this specific NEM instance.

Table 5. Format of NEM Instance Description

Field Name	*Type*	*Description*
Class ID	NEM Spec ID	The identification of the NEM class
Instance ID	Integer	The unique ID provided by the UMF to identify this NEM instance.
Management @	URI	The address of the NEM Management interface.
Acquired Inputs	List<NEMInformation Specification>	Lists the information acquired as inputs by the NEM instance (without the UMF system)
Optional External Inputs	List<NEMInformation Specification>	Lists the information that the NEM instance should receive from KNOW (directly or indirectly)
Mandatory External Inputs	List<NEMInformation Specification>	Lists the information that the NEM instance must receive from KNOW (directly or indirectly)
Available Outputs	List<NEMInformation Specification>	Lists the information that the NEM instance can share with any other UMF entity. This list does not repeat what can be deduced from the other fields of the instance description, i.e. every acquired input can be shared.
Possible Actions	List<NEMAction Specification>	Lists the actions that the NEM instance can apply

Hereafter the reader can find an indicative example of the information comprised in an Instance Description, namely after the creation of the Green TE NEM instance that received the mandate example provided in section 3.1.5.

```
<eu.univerself.nem.InstanceDescription>
    <NEMspecID>
        <Name>Green TE</Name>
        <Provider>StylianosCorp</Provider>
        <Version>1.0.0</Version>
    </NEMspecID>
    <Instance ID>356789456</Instance ID>
    <AcquiredInputs>
        <NEMInfoSpecification>
            <descriptor>Description of router port(ID, capacity)</descriptor>
            <contentType>EthernetPortInfo<!--ID of a ManagementInfoSpec-->
                </contentType>
            <informationUsage>Acquired</informationUsage>
            <type>RawInfo</type>
            <context>{127.100.50.1//all , 127.100.50.5//all , 127.100.50.15//all ,
                127.100.50.19//all , 127.100.50.36//all}</context>
            <controlStatus>Enabled</controlStatus>
        </NEMInfoSpecification>
        <NEMInfoSpecification>
            <descriptor>Description of router interface (ID, capacity, List<Ports
                ID>, IP@)</descriptor>
            <contentType>IPInterfaceInfo<!--ID of a ManagementInfoSpec-->
                </contentType>
            <informationUsage>Acquired</informationUsage>
            <type>RawInfo</type>
            <context>{127.100.50.1//all , 127.100.50.5//all , 127.100.50.15//all ,
                127.100.50.19//all , 127.100.50.36//all}</context>
            <controlStatus>Enabled</controlStatus>
        </NEMInfoSpecification>
        <NEMInfoSpecification>
            <descriptor>Load of router interface</descriptor>
            <contentType>Numeric</contentType>
            <informationUsage>Acquired</informationUsage>
            <type>RawInfo</type>
            <context>{127.100.50.1//all , 127.100.50.5//all , 127.100.50.15//all ,
                127.100.50.19//all , 127.100.50.36//all}</context>
            <controlStatus>Enabled</controlStatus>
        </NEMInfoSpecification>
        <NEMInfoSpecification>
            <descriptor>Routing Table</descriptor>
            <contentType>List&lt;LSA&gt;</contentType>
            <informationUsage>Acquired</informationUsage>
            <type>RawInfo</type>
            <context>{127.100.50.1 , 127.100.50.5 , 127.100.50.15 , 127.100.50.19 ,
               127.100.50.36}</context>
            <controlStatus>Enabled</controlStatus>
        </NEMInfoSpecification>
    </AcquiredInputs>
    <OptionalExternalInputs>
        <NEMInfoSpecification>
            <descriptor>Prediction of router interface load</descriptor>
            <contentType>Numeric</contentType>
            <informationUsage>External Optional</informationUsage>
            <type>Knowledge</type>
            <context>{127.100.50.1//all , 127.100.50.5//all ,
               127.100.50.15//all}</context>
            <controlStatus>Resolved - Enabled<!--Means KNOWLEDGE found an UMF
               entity to provide this knowledge to this NEM instance-->
            </controlStatus>
        </NEMInfoSpecification>
        <NEMInfoSpecification>
            <descriptor>Prediction of router interface load</descriptor>
            <contentType>Numeric</contentType>
            <informationUsage>External Optional</informationUsage>
            <type>Knowledge</type>
            <context>{127.100.50.19//all , 127.100.50.36//all}</context>
            <controlStatus>UnResolved<!--Means KNOWLEDGE did not find an UMF entity
               to provide this knowledge to this NEM instance-->
            </controlStatus>
        </NEMInfoSpecification>
    </OptionalExternalInputs>
    <PossibleActions>
        <NEMActionSpecification>
            <descriptor>Switch ON/OFF Ethernet port</descriptor>
            <contentType>Boolean</contentType>
            <controlFlexibility>{Enabled, Disabled,
               Intercepted}</controlFlexibility>
            <controlStatus>Disabled</controlStatus>
            <target>{127.100.50.1//all , 127.100.50.5//all ,
               127.100.50.15//all}</target>
        </NEMActionSpecification>
        <NEMActionSpecification>
            <descriptor>Switch ON/OFF Ethernet port</descriptor>
            <contentType>Boolean</contentType>
            <controlFlexibility>{Enabled, Disabled,
               Intercepted}</controlFlexibility>
            <controlStatus>Enabled</controlStatus>
            <target>{127.100.50.19//all , 127.100.50.36//all}</target>
        </NEMActionSpecification>
        <NEMActionSpecification>
            <descriptor>Switch ON/OFF IP interface</descriptor>
            <contentType>Boolean</contentType>
            <controlFlexibility>{Enabled, Disabled}</controlFlexibility>
            <controlStatus>Disabled</controlStatus>
            <target>{127.100.50.1//all , 127.100.50.5//all , 127.100.50.15//all ,
               127.100.50.19//all , 127.100.50.36//all}</target>
        </NEMActionSpecification>
        <NEMActionSpecification>
            <descriptor>Change metric of IP interface</descriptor>
            <contentType>Numeric</contentType>
            <controlFlexibility>{Enabled, Disabled,
               Constrained}</controlFlexibility>
            <controlStatus>Disabled</controlStatus>
            <target>{127.100.50.1//all , 127.100.50.5//all , 127.100.50.15//all ,
               127.100.50.19//all , 127.100.50.36//all}</target>
        </NEMActionSpecification>
    </PossibleActions>
</eu.univerself.nem.InstanceDescription>
```

3.1.7 NEM Deletion

A DELETE INSTANCE message is actually a specific case of a NEM INSTANTIATION/DELETION message that follows the format described in Table 2.

3.1.8 NEM's Relations with Coordination

In practice COORD is controlling the NEM to insure orchestration with other NEM instances and avoid conflicts with other NEM instances. The following paragraph details some specific aspects and mechanisms relevant to the control of NEMs by COORD.

First COORD is working with the identification of atomic conflicts between NEMs. This is done by looking at Instance Descriptions of NEMs.

Then COORD is picking conflict avoidance strategies for group of atomic conflicts.

Then COORD is controlling the behaviour of NEMs by enabling and disabling some subsets of the NEMs

In parallel, COORD is applying orchestration policies that drive the way some NEM instances should be triggered after other NEM instances. This will be translated into regime policies (see Figure 6).

Hereafter is a list of refinements that can be provided to the NEM instance descriptions. This paragraph details how some specific sub-classes of either UMFInformationSpec or ActionSpec types can be used to identify specific type of inputs or outputs. Coordination is likely to use this in order to perform conflict avoidance, and also to determine the proper type of conflict avoidance strategy.

- Regarding the possible outputs ((UMFInfoSpecification)) of NEMs, they could make use of the following specific sub-format derived from the Information Model (see Figure 7):
    - utility of a NEM,
    - analytical function of the NEMs' utility,
    - predicted utility,
    - description of an action completed,
    - description of an action to be completed,
    - typical period of the NEM,
    - other (aka generic or undefined)
- Regarding the possible actions produced by NEMs, they could make use of the following specific sub-format derived from the Information Model (see Figure 8):
    - Set parameter value: then specify the parameter name, id (equipment/interface targeted), range
    - Populating a network DB: then specify the DB name, id, and kind of fields that can be populated
    - other (aka generic or undefined)
- Regarding the possible NEMSpecificPolicySpecifications, they could make use of the following specific sub-format derived from the Information Model (see Figure 6):
    - Weighting factor of the utility function, (should provide range or possible values)

To better understand the relation between the NEMs, UMF in general and COORD in particular, it is worth depicting the different time scales at which the NEM is behaving (see Figure 9). The smaller time scale is the time-scale of a cycle of the MAPE autonomic loop of the NEM. Then a bigger time-scale is the period during which COORD is not modifying the control onto the NEM (neither changing the regime nor the action constraints). Then even bigger time-scale is the period during which GOV is setting the NEM in the ready state (see section 3.1.1 Life-cycle of a NEM instance). While the biggest time-scale is the period during which a NEM instance exists. Some of the coordination mechanism will only play with the control of the NEM, while mechanisms like joint optimization are interfering with the NEM at each MAPE cycle (e.g. in order to receive the predicted utility).

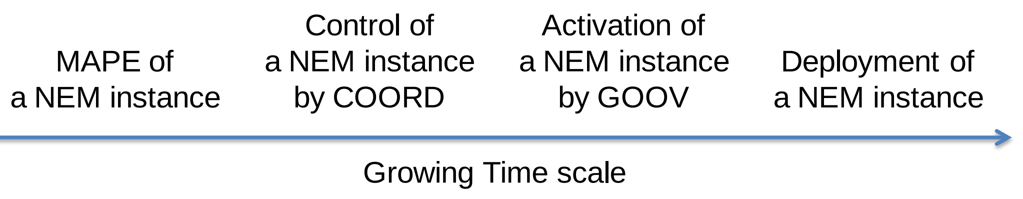

MAPE of a NEM instance
Control of a NEM instance by COORD
Activation of a NEM instance by GOOV
Deployment of a NEM instance
Growing Time scale

Figure 9. Different time scales of a NEM

Table 6 indicates the NEM mechanisms used by the existing coordination strategies (see section 4.3.1 Optimization and conflict avoidance mechanisms):

Table 6. NEM mechanisms used by COORD depending on the coordination strategy.

	Setting runtime regime	NEM providing knowledge	Enabling/ Disabling NEMs actions	Enforcing the action to the NEM	NEM receiving knowledge	Setting range to parameters
Random token	x					
Time separation	x	Typical period				
SOUP	x	Predicted utility	x			
Joint Optimisation	x	Utility,(Utility function)	x	x		
Future strategies?					?	?

Table 7 provides the main lines of an algorithm to determine which coordination strategy to pick depending on the properties of the conflicting NEMs.

Table 7. Criteria for determining which coordination strategy is applicable to which NEM.

Strategy	Conditions on NEM
Random token	Applicable to any NEM
Time separation	Applicable to any NEM that can share its typical period
SOUP	Applicable to NEMs, which are capable of: Providing their predicted utility Enabling/disabling the enforcing of their action
Joint Optimisation	Applicable to NEMs, which are capable of: Providing their utility, Disabling their work, Have a single action, which is of the type set parameter value
Future strategies	Unknown yet

3.1.9 Description of the operations for state transitions

The operations that enable a NEM to go from one state to another state of its lifecycle are described thereafter.

Operation Name	getState

Description	Retrieves the current state of the NEM.
Constraints	The NEM itself must establish "happens-before" relationships between asynchronous operations that change and/or retrieve its state.
List of input data	
List of output data	state : NEMState, the current state of the NEM, the values can be: READY, OPERATIONNAL, DEPLOYING, REGISTERING, UNREGISTERING, UNDEPLOYING, UPDATING, VOID INSTANTIATED
List of non-functional requirements	

Operation Name	getManifest
Description	Retrieves the manifest of NEM class.
Constraints	
List of input data	
List of output data	manifest : NEMManifest, the NEM's manifest (see section 3.1.3)
List of non-functional requirements	

Operation Name	enforceMandate
Description	Sets the mandate for the NEM (see Section 3.1.5). It will validate the addresses of the core blocks contained in the mandate as well as the configuration options to be set, and will return corresponding codes in the case of error. In case a mandate has already been set to the NEM, it will be updated with the new one while any missing fields of the new mandate will be filled-in by the corresponding values of the previous mandate. This operation will trigger the NEM's deployment and registration.
Constraints	Previous mandate has to be already enforced to the NEM in case of missing fields. The NEM has to check option values that require registration change and re-register when necessary. The mandate object might or might not be covering or releasing additional managed equipment.
List of input data	mandate : NEMMandate, the mandate to enforce (see section 3.1.5)
List of output data	result : ActionResult, the result of the operation, containing one of the following codes, as well as AdditionalInfo whenever applicable:

ActionResultCode	Condition/Description
OK	Successful mandate setting.

	INVALID_MANDATE_ADDRESS	One of the CORE addresses specified in the mandate is unreachable. AdditionalInfo in this case specifies which one it is.
	CFG_OPT_NOT_SUPPORTED	The mandate specifies a configuration option not supported by this NEM. AdditionalInfo in this case specifies the missing option's name.
	VALUE_NOT_ALLOWED	The mandate specifies a configuration option value that is not allowed. AdditionalInfo in this case specifies the option's name.
	DEPLOYMENT_ERROR	An error occurred during deployment of the NEM. AdditionalInfo in this case contains debug information.
	REGISTRATION_ERROR*	An error occurred during the registration of the NEM. AdditionalInfo in this case contains debug information.
	*Note that this error code might occur during the re-registration of the NEM because of a configuration option value change that requires re-registration (see ConfigOptionDescription.RequiresRegistrationChange field)	
List of non-functional requirements	After a call to this method the NEM might need to re-deploy and re-register.	

Operation Name	getMandate
Description	Retrieves the mandate that has been set to a NEM using enforceMandate or null if no mandate has been set.

Constraints	The NEM must make sure all the mandate fields regarding configuration options and the managed equipment are up-to-date.
List of input data	
List of output data	NEMMandate (see section 3.1.5)
List of non-functional requirements	

Operation Name	revokeMandate
Description	Revokes any mandate applied to the NEM. This operation will cause the NEM to undeploy and unregister. All subsequent calls to "getMandate" will return null, and the NEM will reach the "VOID_INSTANTIATED" state upon completion of the operation. If the NEM is already in the "VOID_INSTANTIATED" state, this operation has no effect.
Constraints	
List of input data	
List of output data	result : ActionResult, the result of the operation, containing one of the following codes, as well as an "AdditionalInfo" instance whenever applicable:

ActionResultCode	Condition/Description
OK	Successful mandate revocation.
OK_WITH_WARNINGS	Successful mandate revocation but one or more of the interested parties could not be notified (e.g. COORD or KNOW). Additional Info in this case contains debug information.

List of non-functional requirements	Note that there is no reason for which a NEM will not go to the state "VOID_INSTANTIATED" after this operation. Even if, for instance, COORD or KNOW is down and cannot be notified of the NEM's stopping.

Operation Name	setUp
Description	Activates a NEM to start operating.
Constraints	The NEM must be on the "READY" state during a call to setUp. If that is not true during a call to setUp, error "NEM_NOT_READY" is returned.
List of input data	
List of output data	result : ActionResult, the result of the operation,

	containing one of the following codes, as well as AdditionalInfo whenever applicable:

ActionResultCode	Condition/Description
OK	Successful activation.
NEM_NOT_READY	Returned upon a call to activate the NEM while it's not in the "READY" state. AdditionalInfo contains the current state of the NEM.

List of non-functional requirements	

Operation Name	setDown
Description	Deactivates an operating NEM so that it reaches the "READY" state.
Constraints	The NEM might be in any state during a call to setDown. If the NEM is not in the "OPERATIONAL" state during a call to setDown, the operation has no effect, and the current state is returned along with a warning indication in the result.
List of input data	
List of output data	result : ActionResult, the result of the operation, containing one of the following codes, as well as AdditionalInfo whenever applicable:

ActionResultCode	Condition/Description
OK	Successful deactivation.
OK_WITH_WARNING	Returned upon a call to deactivate the NEM while it's not in the "OPERATIONAL" state. AdditionalInfo contains the current state of the NEM.

List of non-functional requirements	

Operation Name	executeControlPolicy
Description	Commands a NEM to execute the specified control policy.
Constraints	The NEM has to be in the "OPERATIONAL" state during this invocation, otherwise an error is returned.
List of input data	policy: ControlPolicy, the policy object to be executed.
List of output data	result : ActionResult, the result of the execution, containing one of the following codes, as well as AdditionalInfo whenever applicable:

ActionResultCode	Condition/Description
OK	Successful execution.
ERROR_NOT_OPERATIONAL	Returned whenever this operation is invoked and the NEM is not in the "OPERATIONAL" state.
OTHER_ERRORS..... OR.....WARNINGS	???

List of non-functional requirements	

Operation Name	delete
Description	Revokes the NEM's mandate and deletes the instance, hence it terminates any process it is running in, thus, releasing any resources associated with the NEM. This method has the exact same effect of "revokeMandate" plus it causes the NEM to terminate after the revocation is completed.
Constraints	
List of input data	
List of output data	(see output of "revokeMandate")
List of non-functional requirements	(see NF requirements of "revokeMandate")

Operation Name	addStateTransitionListener
Description	Subscribes a listener to be notified of state change events.
Constraints	
List of input data	listener: StateTransitionEventListener, the instance to notify of state changes.
List of output data	
List of non-functional requirements	If the listener is already in the list, this operation has no effect.

Operation Name	removeStateTransitionListener
Description	Unsubscribes a listener of state change events.
Constraints	
List of input data	listener: StateTransitionEventListener, the instance to remove from the list of listeners.
List of output data	
List of non-functional requirements	If the listener is not in the list, the operation has no effect.

3.2 Governance block

Governance is a way to control and manage networks that integrate autonomic capabilities. The aim of governance is to allow the human operator to pilot his network through high levels business objectives that is without the need of having deep technical knowledge of the network. Governance also offers an autonomic oriented network and service view to the operator with a two-fold mission: delivering the status of network resources and deployed services, report on the ability of autonomic applications to fulfil the business goals. The provided governance operations can be used to design a “look and feel” Human to Network tool that will enable the operator access in a more intuitive way the network view of its interest.

Alike any other UMF Core Block, GOV is also implementing at least a KnowledgeExchangeInterface in order to receive and provide flows of information under the control of the Information Flow Establishment and Optimisation function of KNOW (see section 3.3.4).

List of Governance block functions:

- Human to Network Interface
- Policy Derivation and Management
- NEM Management
- Enforcement

3.2.1 Human to Network Interface

The Human to Network Interface function provides a friendly way of creating and editing policies using a high level business language. It is the main communication channel between UMF and the human operator.

The main functionality of the H2N interface is to provide a tool for the human operator to insert high-level business objectives, which will be later on translated autonomously into technology-specific terms so that the human operator does not need to deal with any technical details. High level objectives may be related to the introduction of a new application, sets of user classes for the application, sets of Quality of Service (QoS) levels for each user class of the application, etc. These high-level objectives/policies need to be further propagated to the network going through a set of levels (related to different aspects of the management of a communications network) and be transformed into lower level policies so that they reach the element(s) in which to be enforced in terms of low level, technology-specific commands. Consequently, the already set business goals are forwarded to the Policy Derivation & Management block in order be translated from service requirements into network configuration (technology-specific terms) and leave the system to autonomously work out the situation and meet the objectives. The H2N interface also allows feedback, e.g. the result of diagnosis or a visualization of the monitoring to the system administrator/operator.

List of “Human to Network Interface” operations:

High Level Parameters Definition, Service Definition, Network & Service Supervision

Name	High Level Parameters definition
Description	High-level parameters definition block allows the composition of high level parameters for a given service, network operation, group of services or group of network operations. For instance, the human operator can define that Gold users using streaming service should experience excellent levels of availability, reliability, speed and security.
Constraints	
List of input data	Performance parameters
List of output data	Business policies
List of non-functional requirements	

Name	**Service definition**
Description	Service definition allows the specification of operator's parameters:, type of service, network technologies, user classes, available levels of availability, reliability, speed and security, etc.
Constraints	
List of input data	Service attributes
List of output data	Service
List of non-functional requirements	

Name	**Network and Service Supervision**
Description	Network & Service supervision function allows the visualization of the network topology, status and alerts. As deduced from Milestone 25, "Human factors in network management", one of the demands of human operators concerns the supervision of the functioning of the autonomic network, a factor that is closely related to trust. In general, the request was a tool able to provide the information required at the first sight, but with the possibility of getting more detailed information when needed. Tools should also provide trustworthy information and of an appropriate amount. They should also be usable so that there is not much manual work and provide access to all equipment that should be supervised.
Constraints	
List of input data	Network monitoring information : ServiceStatisticalInfo, ResourceStatisticalInfo, Performance, ResourceStateInfo, ServiceStateInfo
List of output data	N/A (visualization of input data)
List of non-functional requirements	• good graphical user interface, which should provide the information required at the first sight, but there should be a possibility to get more detailed information when needed • easy to use

It is worth noting that these operations can be implemented in a dedicated graphical user interface, or alternatively can be implemented as interfaces to the existing OSS and BSS systems of the operator.

3.2.2 Policy Derivation and Management function

The Policy Derivation and Management (PDM) function is in charge of (i) providing facilities for the policies edition and storage (insertion, modification, retrieval and removal of policies) (ii) translating business language to more specific policy language statements, (iii) checking whether the different policies have conflict, (iv) in case conflicts appear, resolve them according to the well-defined conflict resolution mechanisms, and, finally (v) ensuring cohesion between different forms of policies at different levels of abstractions.

Translation is typically done through a set of levels (related to different aspects of the management of a communication network) and produces as its final output a set of lower level policies that can be understood and interpreted by NEMs (the so-called NEM policies). Three translation levels were adopted and defined as follow:

- "Business level" which correspond to "Market, product & customer" of eTOM (policies related to Strategy, Infrastructure and Product (SIP) and Operations (OPS) processes).

- “Service level” which correspond to “Service” of eTOM (policies related to Service management and operations processes of OPS).
- “NEM level” which correspond to “Resource” of eTOM (policies related to Resource management and operations processes of OPS).

The levels in parallel with eTOM business process framework levels are represented in Figure 10.

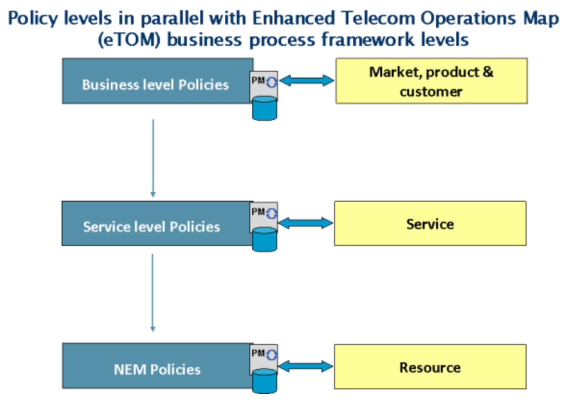

Policy levels in parallel with Enhanced Telecom Operations Map (eTOM) business process framework levels
Business level Policies
PM
Market, product & customer
Service level Policies
PM
Service
NEM Policies
PM
Resource

Figure 10. Policy levels of UniverSelf approach in parallel with eTOM business process framework levels.

As illustrated in Figure 11, the “Business level” policies are technological/administration oriented and technology independent, the “Service level” policies are service oriented and technology independent and the “NEM” policies that are technology dependent. The NEM policies are then enforced onto the corresponding NEMs, which in turn will transform them to device-specific commands (in most cases, vendor specific commands) and enforce them into their managed network elements that belong to any of the network segments. This latest translation is handled by the vendor specific wrappers developed inside the NEMs.

The specification of this number of policy levels enables policy continuum and the operations described above should be performed in each level of the policy continuum. Hence, we suggest an operational layered structure, where each layer corresponds to a level of the Policy Continuum.

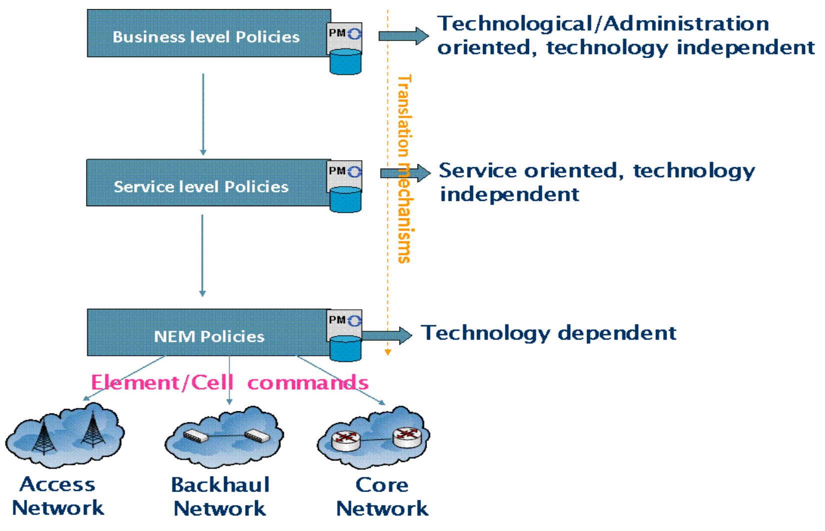


Figure 11. Policy content per level.

Once a policy is created at any of the policy continuum level, it must be analyzed for correctness through a dedicated process (syntactic analysis). Then, the newly created policy should also be analyzed for conflicts detection. If the policy does not conflict with existing policies at the same level, it is translated into policies of the lower level. The outlined process is repeated, until the derivation of NEM policies.

The policies must be expressed using the SID policy model. SID defines Event-Condition-Action (ECA) policies, that is, an Event triggers the invocation of the rule, and if the condition is satisfied, then the action is carried out. Figure 12 and Figure 13 shows the representation of a policy rule and policy structure in UMF. A summary of SID Policy Model is provided in Annex B.

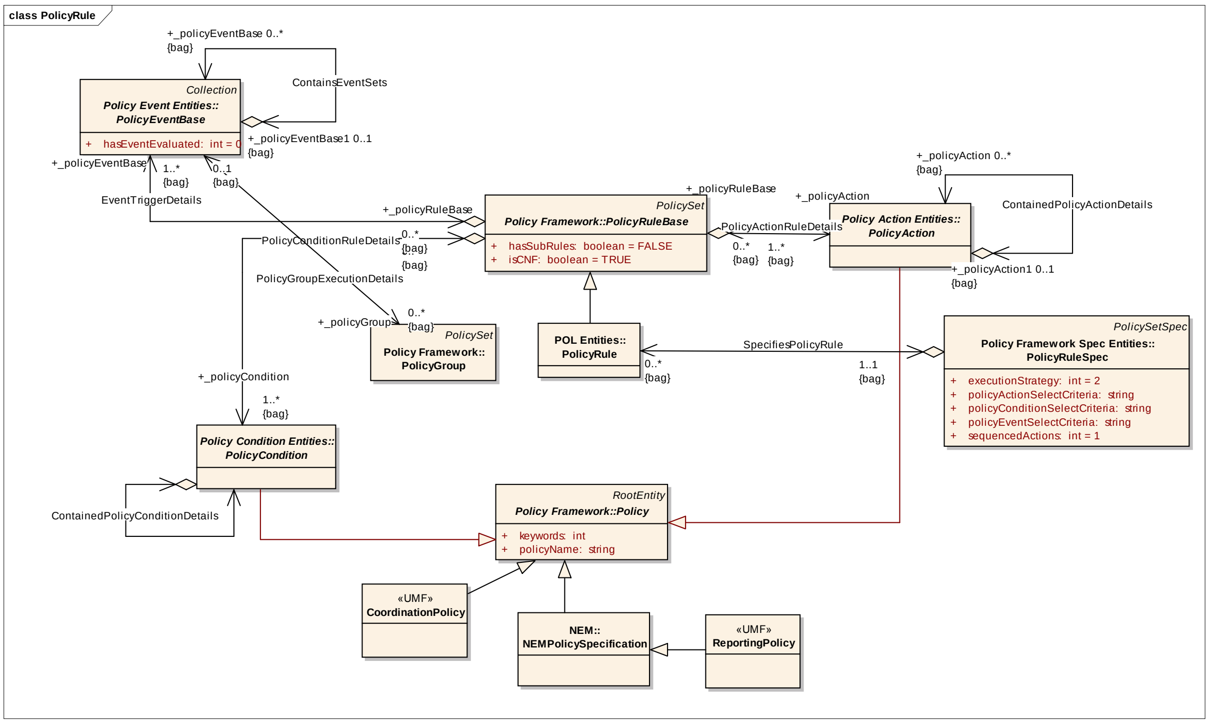

class PolicyRule
Policy Event Entities::PolicyEventBase
Policy Framework::PolicyRuleBase
Policy Action Entities::PolicyAction
Policy Framework::PolicyGroup
POL Entities::PolicyRule
Policy Framework Spec Entities::PolicyRuleSpec
Policy Condition Entities::PolicyCondition
Policy Framework::Policy
«UMF» CoordinationPolicy
NEM::NEMPolicySpecification
«UMF» ReportingPolicy

Figure 12. Representation of a PolicyRule.

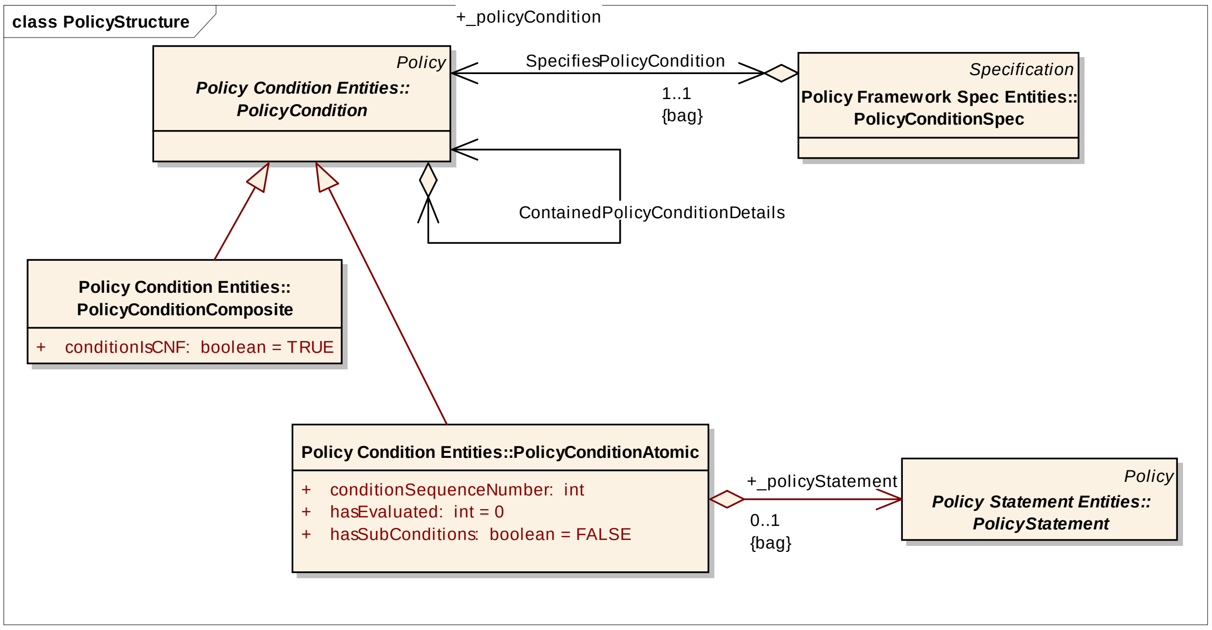

class PolicyStructure
+_policyCondition
SpecifiesPolicyCondition
1..1 {bag}
Policy
Policy Condition Entities::PolicyCondition
Specification
Policy Framework Spec Entities::PolicyConditionSpec
ContainedPolicyConditionDetails
Policy Condition Entities::PolicyConditionComposite
+ conditionIsCNF: boolean = TRUE
Policy Condition Entities::PolicyConditionAtomic
+ conditionSequenceNumber: int
+ hasEvaluated: int = 0
+ hasSubConditions: boolean = FALSE
+_policyStatement
0..1 {bag}
Policy Statement Entities::PolicyStatement

Figure 13. Representation of PolicyStrusture.

List of "Policy Derivation and Management" operations:

Build Business Policy, Create Policy Entry, Retrieve Policy, Update Policy, Delete Policy, Validate Policy, Detect Policies Conflicts, Translate Policy, Check Feasibility & Optimize, Policy Efficiency

Operation 1	**Build Business Policy**

Description	Build Business Policy from High Level Objectives (HLO) or High Level Parameter (HLP)
Constraints	HLP/HLO are provided by BSS operator via a specialised human to network GUI.
List of input data	Performance parameters
List of output data	Business Policy
List of non-functional requirements	N/A

Note: This operation could be realized at the BSS level

Operation 2	**Create policy entry**
Description	Create policy in the policy repository
Constraints	Precondition: policy repository address available.
List of input data	Policy description
List of output data	Notification (Ok/nOK)
List of non-functional requirements	

Operation 3	**Update policy**
Description	Update the policy content
Constraints	Precondition: policy exists in the repository
List of input data	Policy description/format
List of output data	Notification (Ok/nOK)
List of non-functional data	

Operation 4	**Delete policy**
Description	Delete a policy from the policy repository
Constraints	Precondition: policy exists in the repository.
List of input data	NEM ID, Policy ID/policy criteria NOTE: NEM ID is used to delete all its policies Policy criteria: corresponds to research criteria.
List of output data	Notification (Ok/nOK)
List of non-functional requirements	

Operation 5	**Retrieve policy**
Description	Retrieve policy from repository
Constraints	Precondition: policy exists in the repository.
List of input data	NEM ID/Policy ID/Policy criteria NOTE: NEM ID is used to retrieve all its policies Policy criteria: corresponds to search criteria.
List of output data	Policy List that matches the criteria. Empty list if no policy matches the criteria.
List of non-functional requirements	

Operation 6	**Validate policy**
Description	Validate the correctness (in terms of syntax and

	values) of a policy
Constraints	
List of input data	Policy
List of output data	Boolean indicating whether the Policy is syntactically valid or not.
List of non-functional requirements	

Operation 7	**Detect policy conflicts**
Description	Detect conflicts between policies applicable to a NEM
Constraints	Preconditions: more than one policy exists in the Policy repository.
List of input data	Policy list or NEM ID
List of output data	Boolean, Conflicted Policy list
List of non-functional requirements	

Operation 8	**Resolve policy conflicts**
Description	Resolve policy conflicts using the appropriate resolution mechanisms.
Constraints	Precondition: Policy conflicts detection returns a positive value(Boolean=true)
List of input data	Conflicted Policy list
List of output data	Conflict-free Policy list
List of non-functional requirements	

Operation 9	**Translate policy**
Description	Translate business policies to service policies to NEM policies.
Constraints	Precondition: Must be called on a list of conflict-free policies.
List of input data	conflict-free (business or service) Policy list
List of output data	Service policy list or NEM Policy list
List of non-functional requirements	

Operation 10	**Check Feasibility & Optimize**
Description	For each generated policy (business level or service level), it analyses the current status of the network and the available resources, diagnoses potential problems and decides if some kind of optimization should be done for the network to accommodate the requests defined by the policy. For instance, in case the human operator wants to deploy a new service, the Assess Policy Feasibility & Optimize operation is asked to accommodate the request onto the network.
Constraints	
List of input data	List of conflict-free Policies (business level or service level)

List of output data	List of Policies and feasibility report
List of non-functional requirements	

Name	**Policy Efficiency**
Description	Assess the successful translation of high level to low level policies, that is, if the derived policies accomplish the goals described by the operator in the high level policies. A successful policy will lead to well controlled and efficient network operations, while an unsuccessful policy may lead to misconfigurations, QoS / QoE degradation and network instabilities. Thus, a mechanism able to evaluate the policy translation process and measure the gains from the policy application is necessary. The success of a policy in accomplishing the goals described by the operator is in strong relation with the trustworthiness of this specific policy. Trust of policy can be defined as a comparison between the reference behaviour (the behaviour implied in high level policies) and the actual behaviour (based on measurements) of the network after the implementation of the policy.
List of input data	Network monitoring information : ServiceStatisticalInfo, ResourceStatisticalInfo, Performance, ResourceStateInfo, ServiceStateInfo
List of output data	Policy trustworthiness estimation: List of {Policy, Trust index of the policy}
List of non-functional requirements	

Following the previously mentioned layered structure, the Create policy, Validate policy, Detect policy conflicts, Resolve policy conflicts, Translate policy, Check feasibility & Optimize and Policy Efficiency must take place at each of the levels of the policy continuum.

3.2.3 NEM Management

The “NEM Management function” collects and stores in the NEM registry all the management information of the deployed NEMs. It also manages the state transition (including the activation and deactivation of the autonomic control loops) of the NEMs and defines the reporting strategy that meet the operator needs. The reported information is also forwarded to the Policy Derivation and Management function and can therefore be used to trigger more relevant policies given the network on-going situation.

List of “NEM Management” operations:

Create NEM entry, Delete NEM entry, Retrieve NEM information, Update NEM information, Build reporting strategy, Send Reporting strategy, Evaluate Deployed Policies, Create NEM Instance

Operation 11	**Create NEM entry**
Description	Insert INSTANCE ID into the NEM registry
Constraints	Pre-condition: NEM registry address available Post-condition: updated NEM registry Create, Retrieve, Update, Delete (CRUD) operations must be provided by data storage system
List of input data	INSTANCE ID or List of INSTANCE IDs
List of output data	Notification(Ok/nOK)

List of non-functional requirements	

Operation 12	**Delete NEM entry**
Description	Delete NEM entry from the NEM registry and the corresponding NEM information (mandate, instance description)
Constraints	Pre-condition: The NEM to be removed must have been previously stored in the registry CRUD operations must be provided by data storage system
List of input data	INSTANCE ID or List of INSTANCE IDs
List of output data	Notification(ok, Error if NEM not in registry)
List of non-functional requirements	Impact versus dependant NEMs

Operation 13	**Retrieve NEM information**
Description	Get NEM information (report/log) according to reporting strategy.
Constraints	Pre-condition: INSTANCE ID and information previously stored in the registry/database Warning: request size or returned information volume
List of input data	reportingPolicySet or reportingPolicy, INSTANCE ID or List of INSTANCE IDs
List of output data	Content of the reporting strategy according to reportingPolicySet(see Figure 6 page 20)
List of non-functional requirements	Performance aspects wrt to request and returned information size

Operation 14	**Update NEM information (status, mandate)**
Description	Updates the status of a NEM in the NEM registry
Constraints	Pre-condition: The NEM to be updated must have been previously stored in the registry Post condition: updated NEM information(new status or new mandate enforced) Create, Retrieve, Update, Delete operations must be provided by data storage system
List of input data	INSTANCE ID or list of INSTANCE ID
List of output data	Notification(ok, Error if NEM not in registry)
List of non-functional requirements	Impact versus dependant NEMs, Stability issues

Operation 15	**Build reporting strategy(=PolicySet)**
Description	Build/create reporting strategy composed by several reporting policies
Constraints	Policy edition operation must be provided by the PBM system.

List of input data	Reporting policies
List of output data	***Reporting strategy Set<ReportingPolicy>***
List of non-functional requirements	

Operation 16	**Send Reporting Strategy**
Description	Send reporting strategy to the NEM
Constraints	List of reporting strategies contains ALL reporting strategies that the NEM must apply. List of reporting strategies replaces the previous reporting strategies.
List of input data	List of PolicySet, ID or list of INSTANCE IDs
List of output data	***Notification (ok, nok, …)***
List of non-functional requirements	

Operation 17	**Create NEM Instance**
Description	Create a NEM instance in the network elements in which the NEM software is stored
Constraints	The NEM software is stored in the network element or a proxy
List of input data	NEMspecID (see 3.1.4)
List of output data	Notification(OK/NOK)

Operation 17	**Set Up a NEM**
Description	Activates a NEM to start operating (see Section 3.1.9)
Constraints	The NEM must be on the "READY" state during a call to setUp. If that is not true during a call to setUp, error "NEM_NOT_READY" is returned.
List of input data	
List of output data	the result of the operation (OK/NEM_NOT_READY)

Operation 17	**Set Down a NEM**
Description	Deactivates an operating NEM so that it reaches the "READY" state (see Section 3.1.9)
Constraints	The NEM might be in any state during a call to setDown. If the NEM is not in the "OPERATIONAL" state during a call to setDown, the operation has no effect, and the current state is returned along with a warning indication in the result
List of input data	
List of output data	the result of the operation (OK/OK_WITH_WARNING)

3.2.4 Enforcement function

Enforcement encapsulates the communication mechanism between Governance and NEMs. It allows the other functions of the Governance block to be independent of the communication aspects for the interconnection with NEMs. The communication between the GOV functions and NEMs is mainly achieved through the MANDATE object.

List of operations: *Generate NEM Mandate*, *Send NEM Mandate*

Operation 18	**Send NEM Mandate**
Description	Sends a new Mandate to a given NEM. This operation is used for instance to change the activity phase of a given NEM.
Constraints	
List of input data	Mandate (see 3.1.5
List of output data	Notification (OK/NOK)
List of non-functional requirements	

Operation 19	**Generate NEM Mandate**
Description	Generates a new Mandate to a given NEM. Receives a list of policies to be enforced to a NEM, retrieves the mandate from the NEM registry, embeds into it the new policies and enforces it into the corresponding NEM.
Constraints	
List of input data	Policies, NEM ID
List of output data	Mandate (see 3.1.5)
List of non-functional requirements	

The sequent activity diagrams (Figure 14 -16) illustrate GOV operation and interaction with the other UMF core blocks.

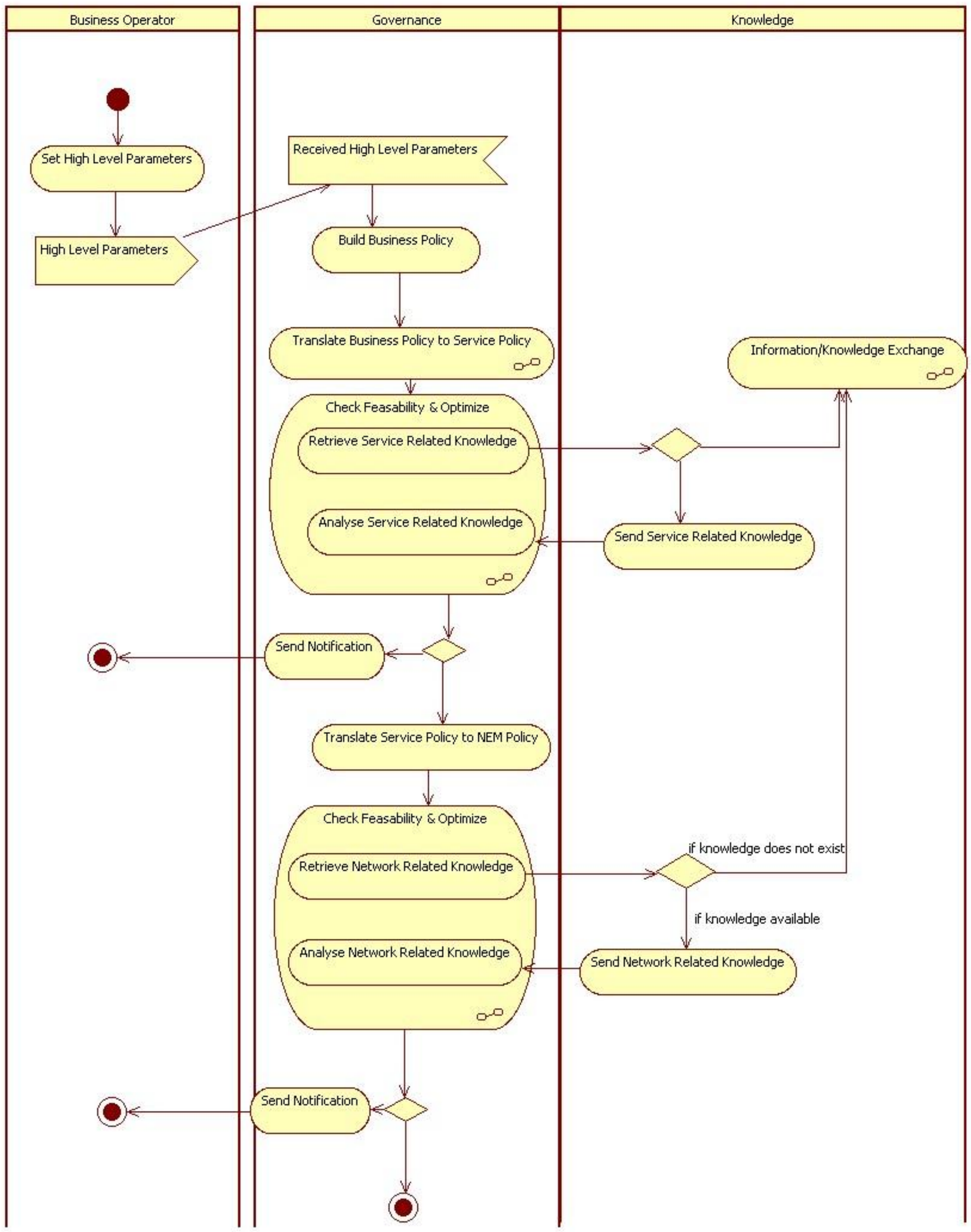

Business Operator
Governance
Knowledge
Set High Level Parameters
High Level Parameters
Received High Level Parameters
Build Business Policy
Translate Business Policy to Service Policy
Information/Knowledge Exchange
Check Feasability & Optimize
Retrieve Service Related Knowledge
Analyse Service Related Knowledge
Send Service Related Knowledge
Send Notification
Translate Service Policy to NEM Policy
Check Feasability & Optimize
Retrieve Network Related Knowledge
if knowledge does not exist
if knowledge available
Analyse Network Related Knowledge
Send Network Related Knowledge
Send Notification

Figure 14. NEM policy definition activity diagram.

In the NEM policy definition activity diagram, the Business Operator sets his high level parameters representing a set of objectives that the network should meet (operation: ***High Level Parameters definition***). These high level parameters are transformed into Business Policies (operation: ***Build Business Policy***). GOV checks the correctness of the policy (*operation: **Validate policy***) and transform the Business policy into Service policy (*operation: **Translate policy***). GOV then controls the correctness of the Service policy (*operation: **Validate policy***), and assesses the feasibility of the new Service policy (operation: ***Check Feasibility & Optimize***). This assessment operation triggers the KNOW block to ask for information about the current status of the network and the available resources (**GOV-KNOW interface)**. When it gets the related information/knowledge, it then analyses the ability of the network to handle the requirements defined in the Service policy. If the control process diagnoses that the service policy cannot be performed, then the GOV sends the appropriate notification to Business Operator with the result of the analysis, the feasibility report. If the control concludes that the Service policy is feasible (probably after the completion of relevant optimization actions from involved entities), then the service policy is translated to NEM policy (*operation: **Translate policy***). After the control of NEM policy correctness (*operation: **Validate policy***), GOV checks again its feasibility (operation: ***Check Feasibility & Optimize***) and request again from KNOW, information about the status of the network and the available resources. In case any of the operations fails, a notification is sent to the human operator. The final outcome of the policy derivation activity is a list of NEM policies.

Figure 15 presents the activity diagram corresponding to the NEM instantiation activity:

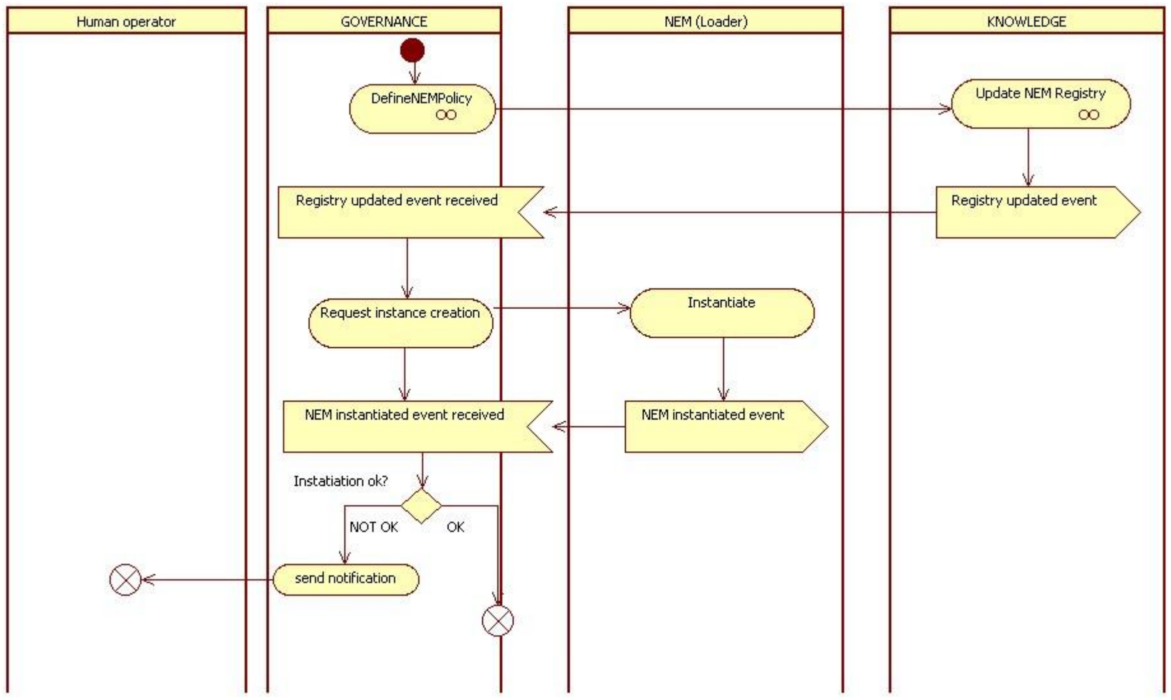


Figure 15. NEM instantiation activity diagram.

For a NEM policy to be deployed and enforced, the corresponding NEMs must have been already instantiated in the network., Figure 15 illustrates the workflow of a NEM instantiation triggered by the action of enforcing a NEM policy. Once a new NEM policy has been defined, it goes into the process of enforcement. Prior to this, the NEM registries are updated with the new information. Then, GOV sends to the corresponding NEM loader the instruction to create an instance (**GOV-NEM interface, operation: Create NEM Instance**). A notification is sent back to the GOV block to report on the action. (***GOV, interface: Send Notification***) (NEM deployment activity diagram) The instantiated NEM is therefore ready to receive and interpret its policies. These policies are used to issue a MANDATE by the "Generate Mandate" and "Send Mandate" operations of the "Enforcement" function and sent to the corresponding NEM.

Figure 16 illustrates the activity diagram of the NEM configuration that can be achieved only through the MANDATE.

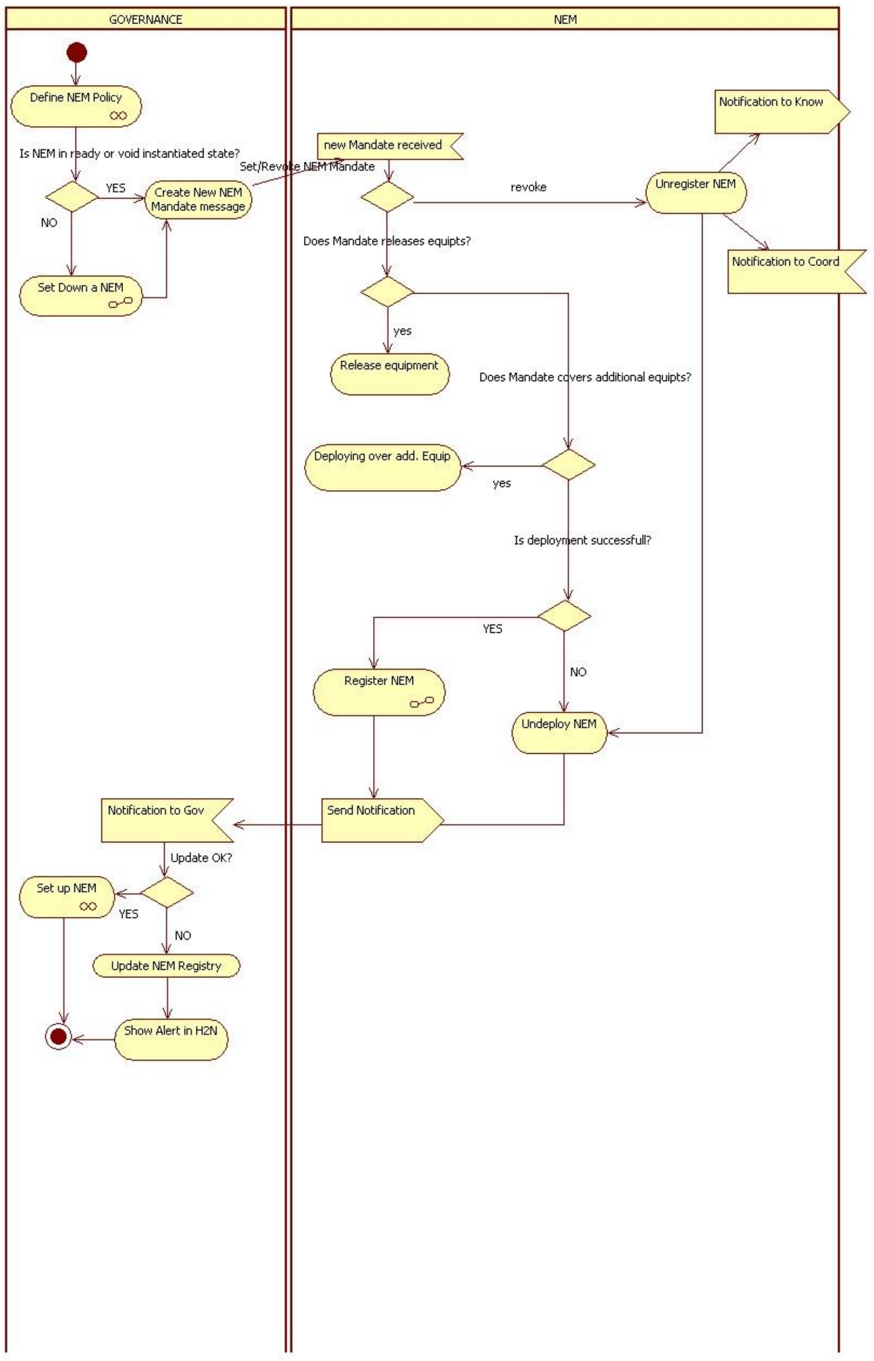


Figure 16. Update Mandate activity diagram.

Once GOV defines/derives a conflict-free NEM policy, it examines if the corresponding NEM is in a ready mode. If it is not, GOV sets down the NEM (***interface GOV-NEM: Set NEM Status***) to bring it in a "READY" mode. When GOV accomplishes this procedure or if the NEM is in ready, then GOV creates new NEM mandate message (operation: ***GenerateNEM mandate***) and sends it to NEM (***interface GOV-NEM: Send NEM mandate***). In case some problem prevents the NEM to self-configure itself according to the mandate, the NEM status in NEM registry is updated (operation: ***Update NEM information***) and a notification is sent to GOV block. If the NEM deployed the new demanded status, then GOV starts up the NEM ((***interface GOV-NEM: Set NEM Status***).

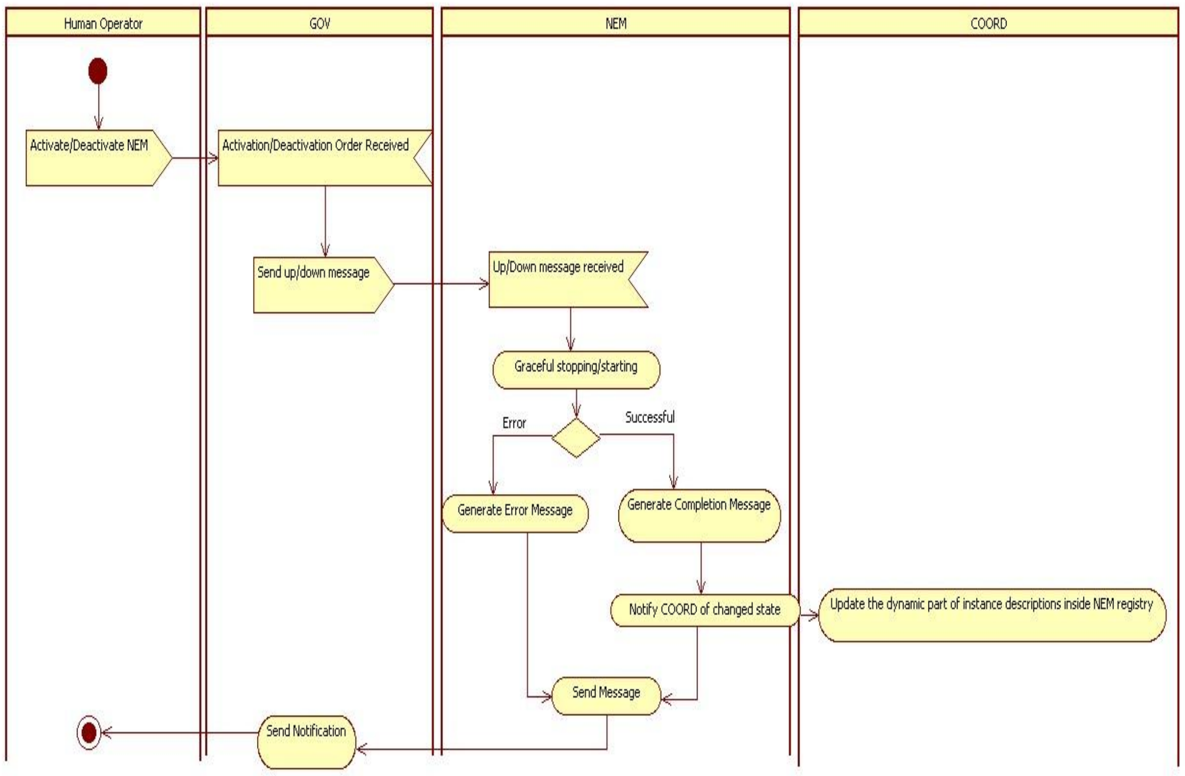

Human Operator
GOV
NEM
COORD
Activate/Deactivate NEM
Activation/Deactivation Order Received
Send up/down message
Up/Down message received
Graceful stopping/starting
Error
Successful
Generate Error Message
Generate Completion Message
Notify COORD of changed state
Update the dynamic part of instance descriptions inside NEM registry
Send Message
Send Notification

Figure 17. Change NEM operational state diagram.

When Human Operator decides that he wants to change the operational state of a NEM (activate/deactivate), he sends the respective command to GOV (***Human operator –GOV interface, operation: Change NEM status***). When GOV receives the command, send to the NEM the respective message, to set its operational state to up/down (***GOV-NEM interface, operation: Set NEM state***).. When the change of the operational state is accomplished, GOV sends the respective notification to human operator.

The following figure depicts the registration phase of a NEM, which just deployed after receiving a Mandate. The NEM is sending its instance descriptions to the interfaces of KNOW, COORD and GOV specified in the Mandate it had received. These UMF core blocks are checking that GOV pre-registered this NEM (to avoid savage NEM deployments). These UMF core blocks are then storing the instance descriptions or at least the information relevant for them, before acknowledging the instance description (see sections 3.1.5 and 3.1.6 for the mandate and instance description formats).

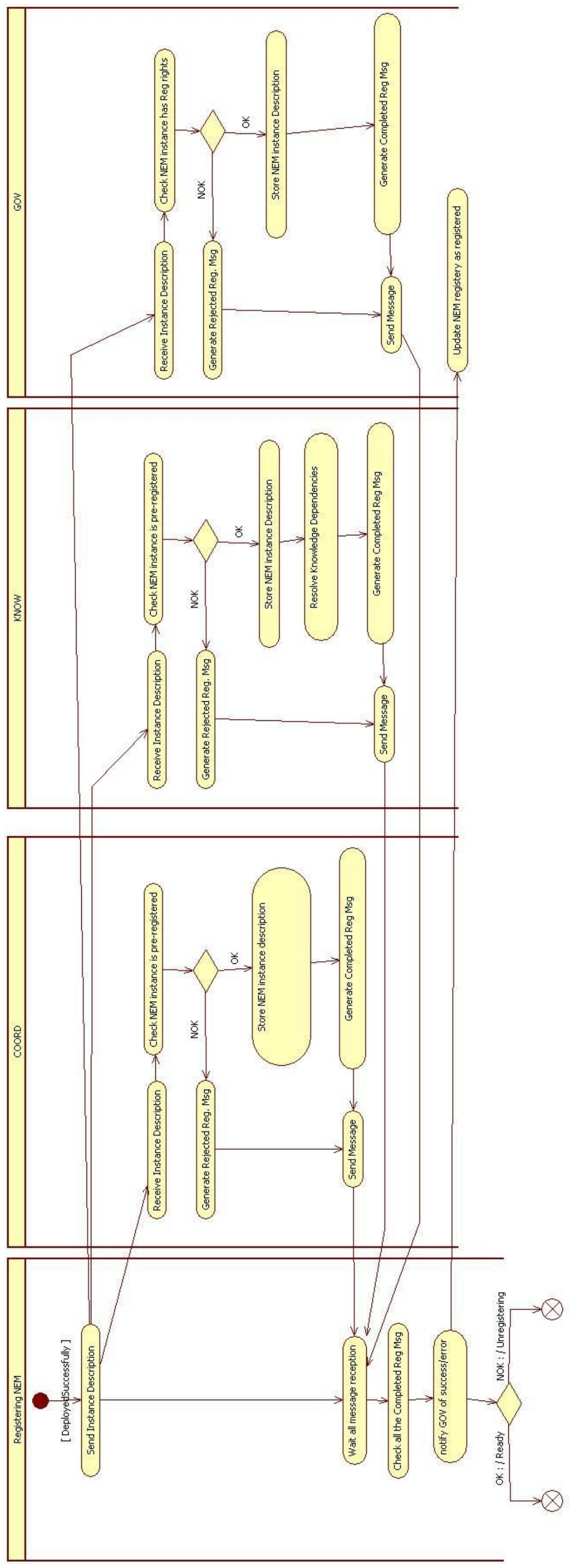

Registering NEM
[DeployedSuccessfully]
Send Instance Description
Wait all message reception
Check all the Completed Reg Msg
notify GOV of success/error
NOK : / Unregistering
OK : / Ready
COORD
Receive Instance Description
Check NEM instance is pre-registered
NOK
OK
Generate Rejected Reg. Msg
Store NEM instance description
Generate Completed Reg Msg
Send Message
KNOW
Receive Instance Description
Check NEM instance is pre-registered
NOK
OK
Generate Rejected Reg. Msg
Store NEM instance Description
Resolve Knowledge Dependencies
Generate Completed Reg Msg
Send Message
GOV
Receive Instance Description
Check NEM instance has Reg rights
NOK
OK
Generate Rejected Reg. Msg
Store NEM instance Description
Generate Completed Reg Msg
Send Message
Update NEM registery as registered

Figure 18. Register NEM activity diagram.

3.3 Knowledge block

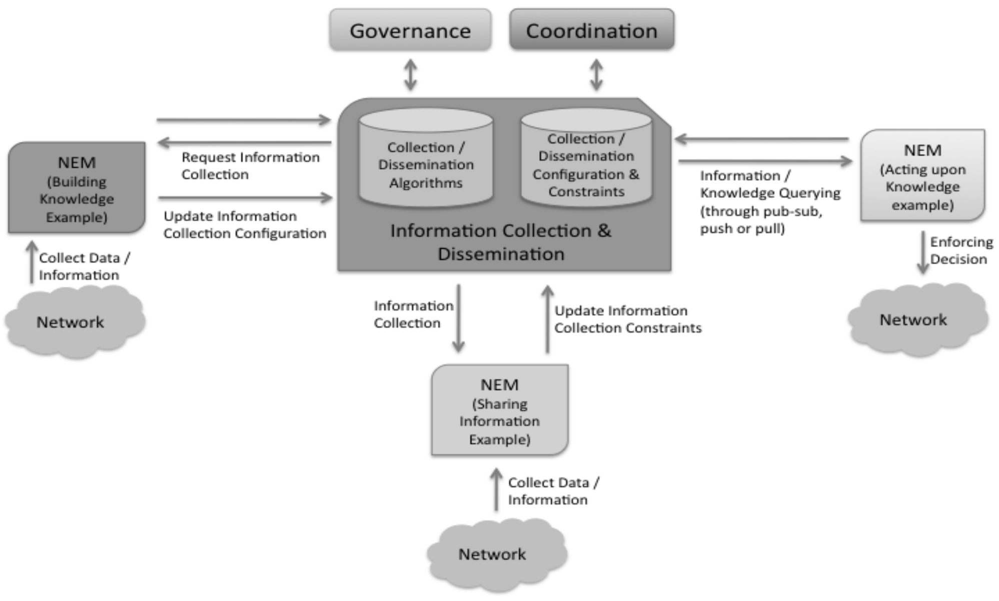
The UMF Knowledge Block (KNOW) is a unified Information and Knowledge Management System. It is a critical part of the UMF since it plays the role of information / knowledge collection, aggregation, storage/registry, knowledge production and distribution across all UMF functional components (i.e., NEMs and Core blocks).

List of Knowledge block functions:

- Information Collection & Dissemination – ICD
- Information Storage and Indexing - ISI
- Information Processing and Knowledge Production - IPKP
- Information Flow Establishment and Optimization - IFEO

3.3.1 Information Collection & Dissemination function

The Information Collection and Dissemination (ICD) function is responsible of information collection, sharing, retrieval and dissemination. An overview of the ICD function is shown in Figure 19.



Governance
Coordination
NEM (Building Knowledge Example)
Request Information Collection
Update Information Collection Configuration
Collection / Dissemination Algorithms
Collection / Dissemination Configuration & Constraints
Information Collection & Dissemination
Information / Knowledge Querying (through pub-sub, push or pull)
NEM (Acting upon Knowledge example)
Enforcing Decision
Collect Data / Information
Network
Information Collection
Update Information Collection Constraints
Network
NEM (Sharing Information Example)
Collect Data / Information
Network

Figure 19. Overview of the Information Collection and Dissemination Function.

The KNOW block handles information from the NEM and eventually form the other Core blocks level and above, produces higher information abstractions and organizes communication of information and knowledge. The ICD function is the front-end of the KNOW block, handling all information exchange between the UMF functional components namely the GOV and COOR core blocks and the NEMs.

These functional components are acting as sources or sinks of information. The sources subscribe to ICD by exposing which type of information they will produced and also the requirements that are associated to this information (frequency, …). On the one hand, each information source should subscribe information availability and the equivalent collection constraints (e.g., the supported granularity of collection). On the other hand, each information sink should subscribe information retrieval requirements with a similar process. The subscription process takes place during the NEM registration (or update) level and is elaborated in the information subscription workflow diagram. The matching of constraints with requirements takes place during an equivalent negotiation process, part of the Information Flow Establishment and Optimization function, elaborated later. The focus of the KNOW block on high information abstractions allows proactive information flow configurations, ready to be instantiated whenever the UMF components need them.

Information can be directly retrieved from or shared with the KNOW block from a UMF component using the appropriate interface (i.e., the Knowledge Exchange Interface). Furthermore, an information collection process is triggered from a component requesting the information, through the ICD function and using the same interface. Then the ICD function will have to collect such information (i.e., pre-processed or not) from the Information Storage and Indexing function (if a historical value of it is present), or it may have to request a suitable Knowledge building NEM to provide such a forecast, and then that particular knowledge building NEM will in turn request the network and/or the KNOW for furnishing the values of specific context data. The information collection process is optimized by the Information Flow Establishment & Optimization (IFEO) function, by e.g., the latter defining filtering objectives and setting appropriate accuracy objectives to be followed by ICD; more details are given in the respective function description. Example information exchanges using the pull and pub-sub methods are illustrated in the knowledge exchange workflows in the end of the section.

The collected information may either be directed to the Information Processing and Knowledge Production function for a further processing (e.g., aggregation) and then indexed / optionally stored to the Information Storage and Indexing function or indexed / optionally stored directly to the latter function. The storage option within Information Storage and Indexing may be provided or demanded based on the nature of the information, NEM demands, optimization goals, etc. After this stage, the information or produced knowledge could be passed back to the ICD function for dissemination. More details on the interactions between the ICD and the ISI function can be found in the next subsection.

The ICD KNOW function supports a Redirect mode. The basic information communication mode between NEMs and KNOW imply a proxy-mode of interaction, where all interactions are handled via the KNOW. However, we also specify a Redirect mode where the KNOW can redirect information querying NEMs towards the appropriate resources (or other NEMs) for direct interaction (i.e., bypassing the KNOW). This process may involve the Coordination block too. The interested NEMs (via the Coordination block or not) may request such direct communication from the KNOW. Another approach is to allow the IFEO function to enforce such direct interactions transparently, for optimization purposes.

All ICD algorithms (e.g., for collection or dissemination) can be configured from the Governance block. This process gives flexibility to the KNOW infrastructure to meet new information manipulation demands, as soon as they arrive. At this stage of work, we explore a number of algorithms or methods (i.e., discussed in the ICD mechanisms subsection) but the selection and grouping of methods to specific contexts will be considered in the last version of the deliverable.

List of "Information Collection and Dissemination" operations:

Information collection, Information sharing, Information retrieval, Information dissemination

Operation 1	**Information collection**
Description	The KNOW block is collecting information from a number of NEMs according to the information collection requirements and should meet certain information collection constraints. Information collection could be one of four types: (i) **1-time queries**, which collect information that can be considered static, e.g., the number of CPUs, (ii) **N-time queries**, which collect information periodically for a certain number of times, (iii) **continuous queries** that collect information in an on-going manner, and (iv) **unsolicited acquisition** of subscribed information units. Information collection is triggered from the KNOW block, in response of an information retrieval request (in case the requested information is not available in the storage).
Constraints	The NEMs sharing information with the KNOW, set the information collection constraints during their registration phase or update the constraints during a

	NEM configuration update phase (e.g., to respond to a network event like congestion). The same process enables the corresponding NEMs as information sources.
List of input data	The NEM or set of NEMs producing the information, id or type of information to be collected.
List of output data	Information
List of non-functional requirements	N/A

Operation 2	**Information sharing**
Description	An information source may share information to the KNOW block (or update existing information). The information sharing is triggered from the information sources.
Constraints	The NEMs sharing information with the KNOW, set the information collection constraints during their registration phase or update the constraints during a NEM configuration update. The same process enables the corresponding NEMs as information sources.
List of input data	Id or type of information, the information
List of output data	ACK
List of non-functional requirements	N/A

Operation 3	**Information retrieval**
Description	Information can be queried from the Knowledge Building NEMs during their knowledge building process and both information & knowledge can be queried from the Actor NEMs that perform optimization or configuration changes. These interactions are handled through the Knowledge Exchange Interface. The information retrieval operation may use the same methods with the information collection operation.
Constraints	Information is available in the KNOW storage or indexed. The information sinks should register their information requirements during their NEM registration or configuration update.
List of input data	Id or type of information requested, further processing requirements (e.g., the need of aggregation etc.).
List of output data	Information, aggregated information or knowledge
List of non-functional requirements	N/A

Operation 4	**Information dissemination**
Description	The ICD function performs information dissemination to a number of NEMs that build knowledge or that act upon this information, e.g., performing configuration changes. The information/knowledge is disseminated using one of the following methods:

	• **Push method:** The KNOW responds to a single information push request coming from a NEM using the Push method. The Information & Knowledge Dissemination block periodically pushes updated information to the interested NEMs (i.e., whenever it changes). The NEMs maintain the information in a local storage, from which they service either knowledge production or act upon the new information; • **Pull method:** NEMs may request information/knowledge using the Pull method. The NEMs must explicitly request a particular type of information and/or knowledge. They can either make these requests on a periodic basis (polling) or when a certain demand arises. An example workflow diagram on the pull method is illustrated in Figure 23 page 62. • **Pub/sub method:** The NEMs can be subscribed to receive a certain type of information and/or knowledge. They are automatically informed when this information appears or changes (e.g., a change higher than a particular threshold). An example workflow diagram on the pub-sub method is illustrated in Figure 24.
Constraints	The information is available and information sinks have subscribed their equivalent information requirements during their NEM registration or configuration update.
List of input data	Id or type of information, dissemination method
List of output data	Information
List of non-functional data	N/A

3.3.2 Information Storage & Indexing function

The Information Storage and Indexing (ISI) function is a logical construct representing a distributed repository for registering NEMs, indexing (and optionally storing) information/knowledge. An overview of the ISI function is shown in Figure 20. The ISI function stores information ranging from NEM registration information to (optionally) knowledge. The ISI functionality includes methods & functions for keeping track of information sources, including information registration and naming, constraints of information sources, information directory and indexing. An important storage aspect, which can assist the knowledge production handled by the Information Processing and Knowledge Production function, is the inherent support of historical capabilities. For example, a NEM could request information and/or knowledge that was stored in the past using an appropriate time stamp. It should be noted that knowledge production functionality is not part of the ISI function, but it supports the storing of knowledge derived due to some earlier calculations. The ISI optionally stores knowledge produced from the Information Processing and Knowledge Production function (for globally-scoped knowledge) or Knowledge Building NEMs (for locally-scoped knowledge).

The different NEMs, either requesting or storing information to the KNOW block, do not directly communicate with the ISI. The ICD function handles information collection or dissemination between the storage points and the NEMs.

The Governance block may parameterize the ISI function, based on the current conditions or global performance goals. For example, the information storage may choose to have alternative structures or configurations (e.g., number of storage nodes, in case of a distributed storage) that are suitable to a particular environment or network condition (e.g., the presence of congestion in the network).

Other important requirements of the ISI function are:

- To be aligned to a pub/sub information dissemination capability.
- To support alternative storage structures, such as centralized/ distributed (flat)/ distributed (hierarchical), based on the context.
- To support information and knowledge caching.

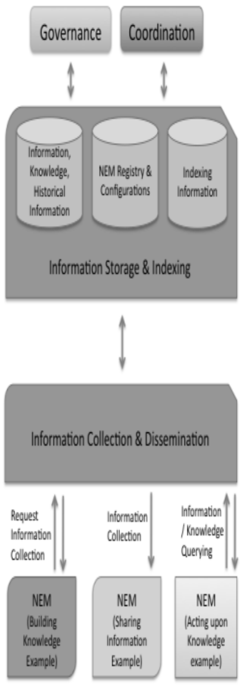

Governance
Coordination
Information, Knowledge, Historical Information
NEM Registry & Configurations
Indexing Information
Information Storage & Indexing
Information Collection & Dissemination
Request Information Collection
Information Collection
Information / Knowledge Querying
NEM (Building Knowledge Example)
NEM (Sharing Information Example)
NEM (Acting upon Knowledge example)

Figure 20. Overview of the Information Storage and Indexing Function.

List of "Information Storage and Indexing" operations:

Information storage, Information indexing, NEM registration

Operation 1	**Store information**
Description	The collected/shared information from/through the ICD function is optionally stored in the Information Storage. After this stage, the information could be passed back to the ICD function for dissemination. Information could be alternatively stored after the end of an information aggregation or knowledge production operation. In case the information is requested through an information retrieval operation, it is fetched from the storage and communicated to the requesting NEM through the ICD.

Constraints	The type of information to be stored should be defined beforehand during a NEM registration phase.
List of input data	List<UMFInformation>
List of output data	ACK
List of non-functional requirements	N/A

Operation 2	**Index information**
Description	Information communicated through the ICD function is optionally indexed. After this stage, the information could be retrieved and passed back to the ICD function for dissemination. Indexed information could be collected as part of an information aggregation or knowledge production operation
Constraints	The type of information to be stored should be defined beforehand during a NEM registration phase.
List of input data	List<UMFInformationSpecification> (see Figure 7 page 21)
List of output data	NEM Instance ID or UMF CORE BLOCK ID for the location of the information (for an index request)
List of non-functional requirements	N/A

Operation 3	**Register NEM**
Description	All NEMs should be registered to the KNOW block. This process includes their information requirements and capabilities. The ISI function maintains a NEM registry, including specifications for the available information to be collected, retrieved or disseminated.
Constraints	In case the NEM is already registered, the NEM information is updated.
List of input data	NEM Instance Description (see section 3.1.6)
List of output data	ACK
List of non-functional requirements	N/A

3.3.3 Information Processing & Knowledge Production function

The Information Processing and Knowledge Production function (IPKP) is responsible for operations related to information processing (e.g., aggregation) and knowledge production. In Figure 21 we show an example diagram of the function and its basic interactions with other KNOW functions. We detail below the different operations of the IPKP function.

A central operation of information processing is the information aggregation (IA). The IA can receive the collected data from the ICD function and filter them out before they are stored to the ISI function or disseminated. Again, this reduces the volume of measurements by only sending values that are significantly different from previous measurements. Using filtering in this way in the KNOW block, lowers the load of the management network. Furthermore, the IA component itself can be flexible enough to be given different aggregation specifications by the Governance block in order to process the data in a varying way. For example, it can be configured to wake up once an hour and select data for the last day, and then apply an aggregation function. This can be achieved using a mechanism that relies on plugins.

As well as requesting information, a NEM may subscribe to an event-based notification service (i.e., a pub/sub mechanism) by setting an appropriate threshold to a specific type of information. Whenever this threshold is exceeded, the subscribed NEM is notified. Aggregation is done during the information collection phase, in order to minimize overhead.

Accordingly, Knowledge Production (KP) component handles and produces globally-scoped knowledge. This type of knowledge is being produced out of aggregated information or locally-scoped knowledge. Locally-scoped knowledge, on the other hand, is built from the Knowledge Building NEMs out of data/information directly collected from the managed entities. In both cases, reasoning and inference mechanisms are required. Thus, similar software components can be used.

These software components can be based on a number of different techniques depending on the exact problem that is addressed, the type of inputs that are used and the type of output that needs to be acquired. Such techniques may come from scientific areas like statistics, clustering, reasoning, Fuzzy or machine learning (including supervised, unsupervised and reinforcement learning techniques).

The produced knowledge from the IPKP block can be optionally stored in the ISI function so as to be available for UMF core mechanisms or NEMs when requested/needed.

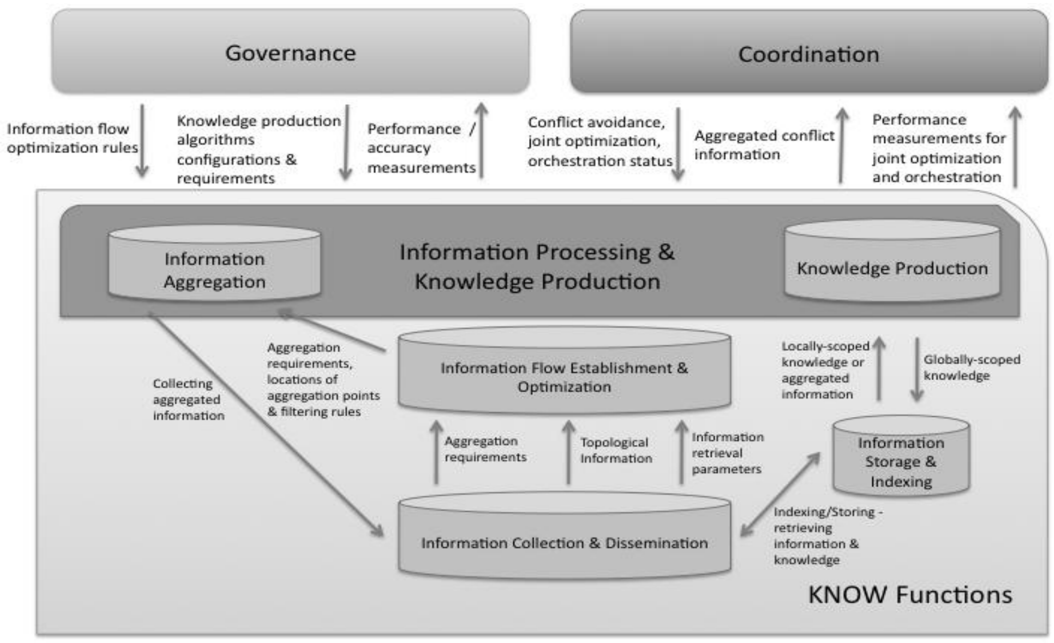

Governance
Coordination
Information flow optimization rules
Knowledge production algorithms configurations & requirements
Performance / accuracy measurements
Conflict avoidance, joint optimization, orchestration status
Aggregated conflict information
Performance measurements for joint optimization and orchestration
Information Aggregation
Information Processing & Knowledge Production
Knowledge Production
Collecting aggregated information
Aggregation requirements, locations of aggregation points & filtering rules
Information Flow Establishment & Optimization
Locally-scoped knowledge or aggregated information
Globally-scoped knowledge
Aggregation requirements
Topological Information
Information retrieval parameters
Information Storage & Indexing
Information Collection & Dissemination
Indexing/Storing - retrieving information & knowledge
KNOW Functions

Figure 21. Overview of the Information Processing and Knowledge Production Function.

List of "Information Processing and Knowledge Production" operations:

Information aggregation, Knowledge production

Operation 1	**Aggregate Information**
Description	applies aggregation functions to the collected data / information. The aggregation process increases the level of information abstraction, thereby transforming the data into a structured form, but at the same time reducing the load on the network. Aggregation works in situations where NEMs do not need a continuous stream of data from the KNOW, but can get by with an approximation of the data values. For example, getting an occasional

	measurement with the average of the volume of traffic on a network link may be enough for some NEMs. Some common aggregation functions include SUM, AVERAGE, STDDEV, MIN and MAX. Although it is most common to use aggregation functions such as the above, arbitrary functions can be passed in, which give considerable power and flexibility when determining aggregations. For example: a customized function that is more complicated compared to the basic aggregation functions.
Constraints	The information to be aggregated should be in the storage or can be collected at real-time. The latter triggers an information collection operation.
List of input data	Id or type of information, information
List of output data	UMFInformation
List of non-functional requirements	N/A

Operation 2	**Produce/Build Knowledge**
Description	Produces globally-scoped knowledge out of aggregated information or locally-scoped knowledge. Reasoning and inference mechanisms are required for this process.
Constraints	The required aggregated information or locally-scoped knowledge should be available in the storage or can be produced at real-time. The latter triggers an information collection operation.
List of input data	Aggregated information or locally-scoped knowledge
List of output data	Globally-scoped knowledge
List of non-functional requirements	N/A

3.3.4 Information Flow Establishment and Optimisation function

The Information Flow Establishment & Optimization (IFEO) function (see Figure 22) regulates the information flow based on the current state and the locations of the participating components (e.g., the NEMs producing information). In particular, it controls information collection handled from the ICD function, information aggregation in the IA operation, and aggregation node placement. Furthermore, it guides a filtering system for information collection and aggregation points that can significantly reduce the communication overhead. However, the reduction depends on the nature of the metric to be monitored.

Both Information dissemination and collection processes should meet certain information collection/dissemination constraints, being communicated to the KNOW during the NEM registration processes. For example, a number of NEMs may trade information accuracy for communication cost. Such accuracy objectives should also meet performance requirements coming from the Governance block (harmonizing information flow to the global performance goals). The Information Flow Establishment & Optimization function is responsible for such quality enforcing functionalities. In the IFEO function, a negotiation takes place that matches interests with constraints. The outcome of the negotiation is the parameters of the information flow, harmonized with the capabilities of the information sources, the requirements of the information/knowledge sinks and the global performance goals of the Governance block.

The IFEO function controls the KNOW and enforces decisions by communicating with the appropriate KNOW nodes (i.e., in case of a distributed deployment), handled from the corresponding KNOW functions, in order to satisfy the performance optimization requirements (coming from the GOV):

- guides the optimal placement of the KNOW nodes

- regulates the information filtering of information collection and dissemination (i.e., handled from the ICD function), based on accuracy objectives

The above processes are part of the quality enforcement functionality of the KNOW block and all corresponding decisions are being taken from the Information Quality Controller (IQC) Component of the IFEO function.

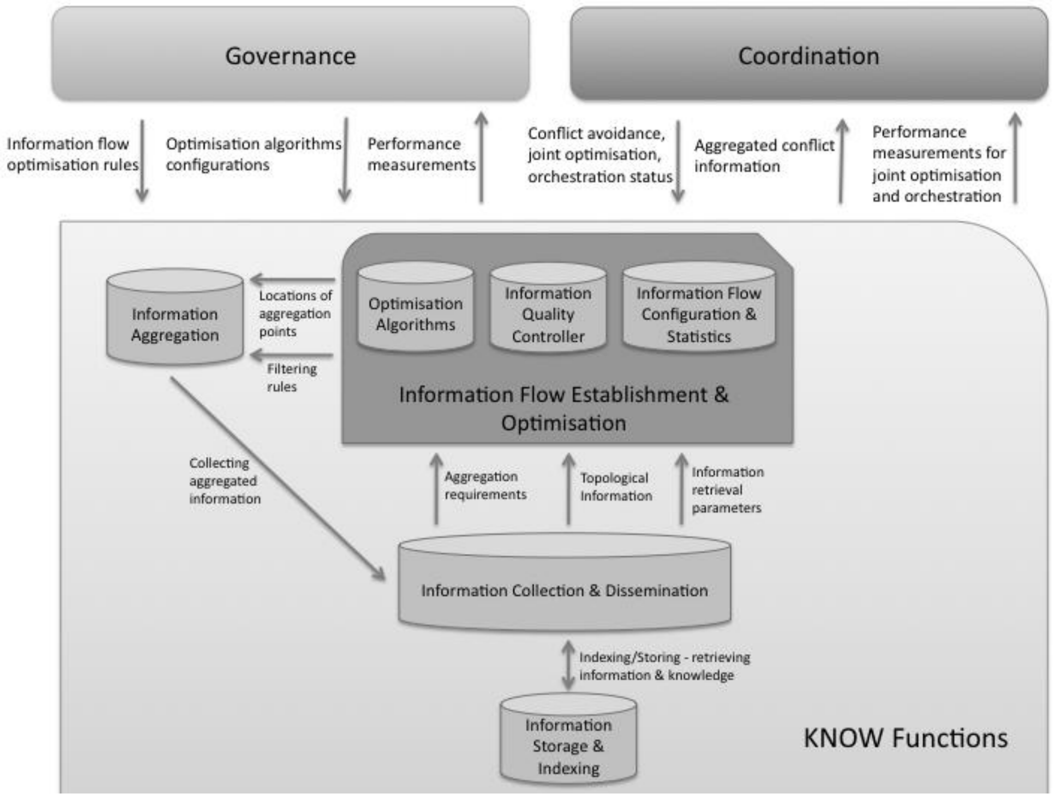

Governance
Coordination
Information flow optimisation rules
Optimisation algorithms configurations
Performance measurements
Conflict avoidance, joint optimisation, orchestration status
Aggregated conflict information
Performance measurements for joint optimisation and orchestration
Information Aggregation
Locations of aggregation points
Filtering rules
Optimisation Algorithms
Information Quality Controller
Information Flow Configuration & Statistics
Information Flow Establishment & Optimisation
Collecting aggregated information
Aggregation requirements
Topological Information
Information retrieval parameters
Information Collection & Dissemination
Indexing/Storing - retrieving information & knowledge
Information Storage & Indexing
KNOW Functions

Figure 22. Overview of the Information Flow Establishment and Optimization Function.

Finally in order to support the indexing and retrieving of information from the indexing table, it is worth mentioning that future work on the UMF should identify/define an ontology to express the relation between network entities and to ensure consistency between the indexed UMFInformationSpecifications (see Figure 7), related also to the section 3.3.2 on Information Storage & Indexing functionon).

List of "Information Flow Establishment and Optimization" operations:

Information flow establishment, Information flow optimization, Information quality control

Operation 1	**Negotiate Information flow**
Description	Responsible for the establishment of every flow of information. This process takes place proactively during a NEM registration or configuration update phase. The flow is optimized from the information flow optimization operation.
Constraints	The information flow constraints come from the NEMs during their NEM registration phase. The constraints should be aligned with the high-level objectives coming from the GOV block.
List of input data	UMFInformationSpecification (see section 3.1.2) and FlowParameters (like Knowledge Excg=hangeInterface URL…)

List of output data	InformationExchangePolicies
List of non-functional requirements	N/A

Operation 2	**Optimize Information flow**
Description	Optimizes each information flow, attempting to meet the expressed constraints and requirements.
Constraints	The information flow optimization operation applies the optimization decisions coming from the information quality controller operation.
List of input data	UMFInformationSpecification, and FlowParameters
List of output data	InformationExchangePolicies
List of non-functional requirements	N/A

Operation 3	**Information quality controller**
Description	Responsible for information flow optimization decisions.
Constraints	The information flow constraints come from the NEMs during their NEM registration (or configuration update) phases. The constraints should be aligned with the high-level objectives coming from the GOV block.
List of input data	UMFInformationSpecification,, information flow constraints, information flow requirements
List of output data	Optimization rules (e.g., filtering based on a certain accuracy objective, optimal number and placement of distributed KNOW nodes etc).
List of non-functional requirements	N/A

Figure 23 illustrates the knowledge exchange process using the pull method. The pub-sub method is illustrated in Figure 24 (publish part). The information subscription diagram illustrated in Figure 25 is part of the NEM registration (or configuration update) operation or it can be part of the configuration of a coordination mechanism.

It is important to note that, the workflows presented there have actors that are named knowledge sink and knowledge source. This means these workflows are characterizing the behaviour (Figure 23 and Figure 24) and the configuration (Figure 25) of a Knowledge Exchange Interface. (see Figure 7 , which depicts the Information model of the UMF information).

This Knowledge Exchange Interface is available at least once in each UMF entity (any of the UMF Core Block and any NEM). The interface is used to exchange knowledge between UMF entities directly or through KNOW which can act as an intermediate repository (see section 3.3.2 Information Storage & Indexing function). In fine the role of the Information Flow Establishment & Optimization function is to organize such flows between UMF entities and to keep track of the existence of these flows.

.

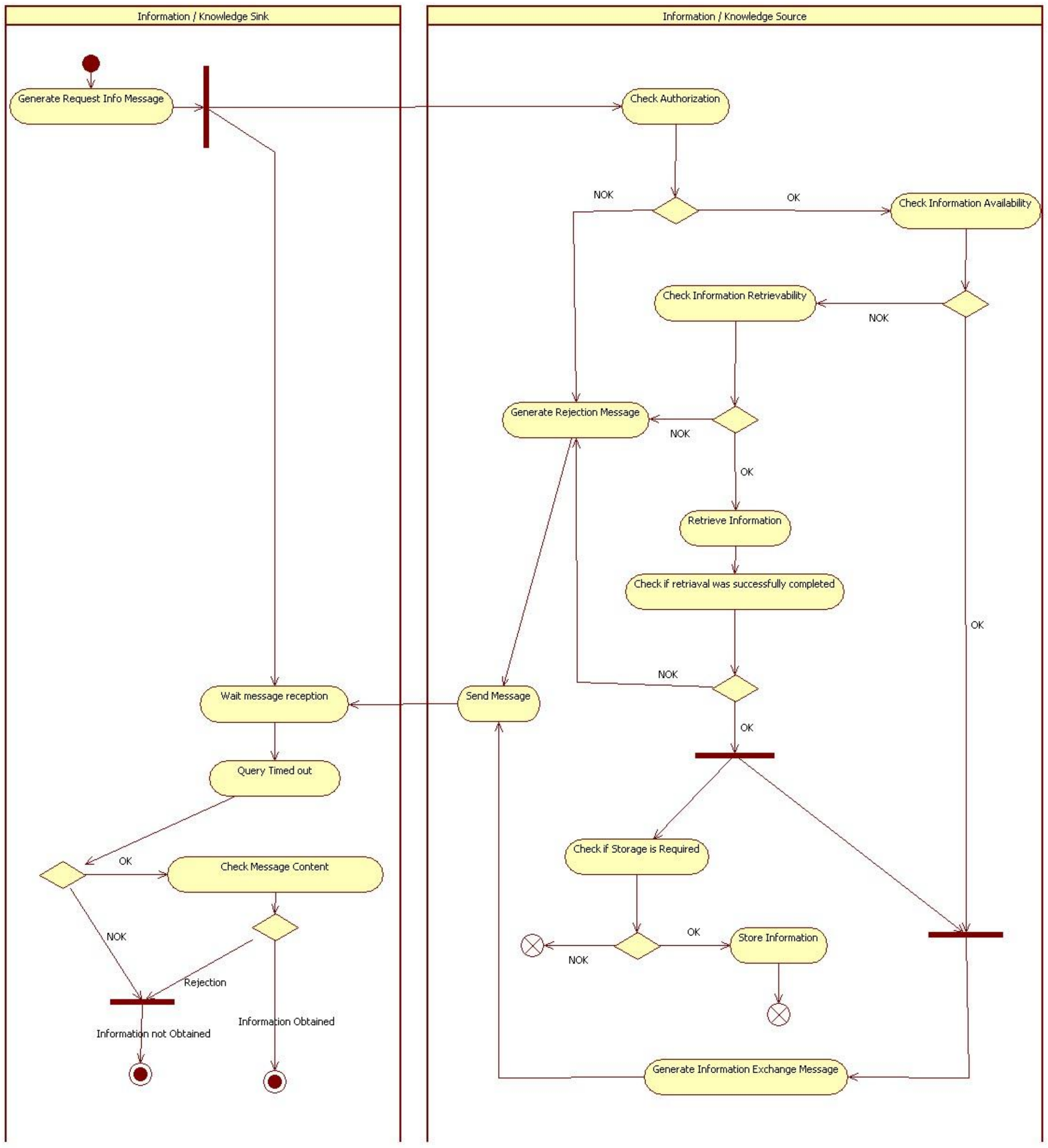

Information / Knowledge Sink
Information / Knowledge Source
Generate Request Info Message
Check Authorization
NOK
OK
Check Information Availability
Check Information Retrievability
NOK
Generate Rejection Message
NOK
OK
Retrieve Information
Check if retrieval was successfully completed
OK
NOK
Wait message reception
Send Message
OK
Query Timed out
Check if Storage is Required
OK
Check Message Content
OK
Store Information
NOK
NOK
Rejection
Information Obtained
Information not Obtained
Generate Information Exchange Message

Figure 23. Knowledge Exchange workflow diagram using the Pull method.

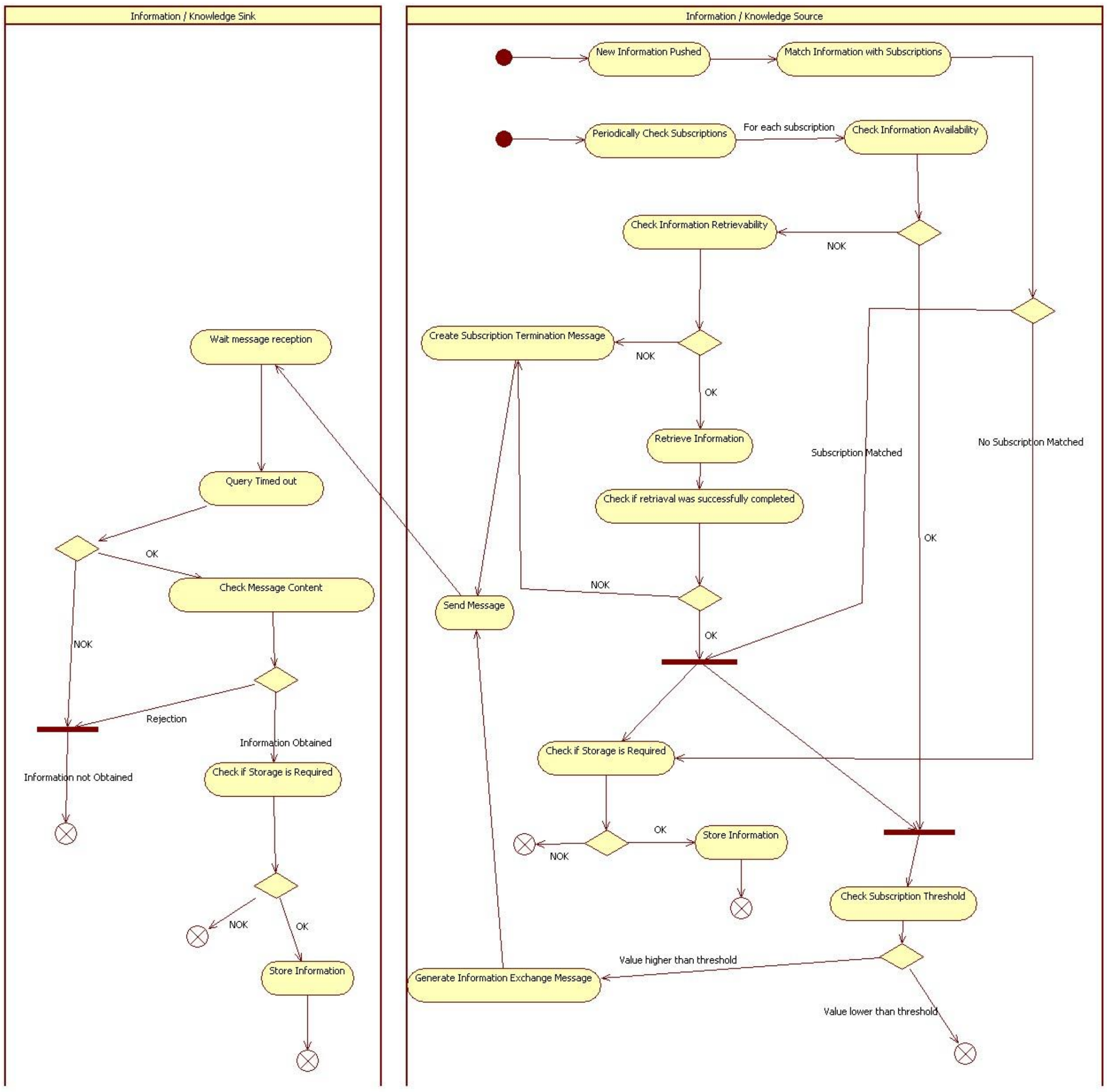

Information / Knowledge Sink
Information / Knowledge Source
New Information Pushed
Match Information with Subscriptions
Periodically Check Subscriptions
For each subscription
Check Information Availability
Check Information Retrievability
NOK
Create Subscription Termination Message
NOK
OK
Wait message reception
Retrieve Information
Subscription Matched
No Subscription Matched
Check if retriaval was successfully completed
OK
Query Timed out
OK
Check Message Content
Send Message
NOK
OK
NOK
Rejection
Information Obtained
Check if Storage is Required
Information not Obtained
Check if Storage is Required
OK
Store Information
NOK
Check Subscription Threshold
NOK
OK
Store Information
Generate Information Exchange Message
Value higher than threshold
Value lower than threshold

Figure 24. Knowledge Exchange workflow diagram using the Pub-sub method.

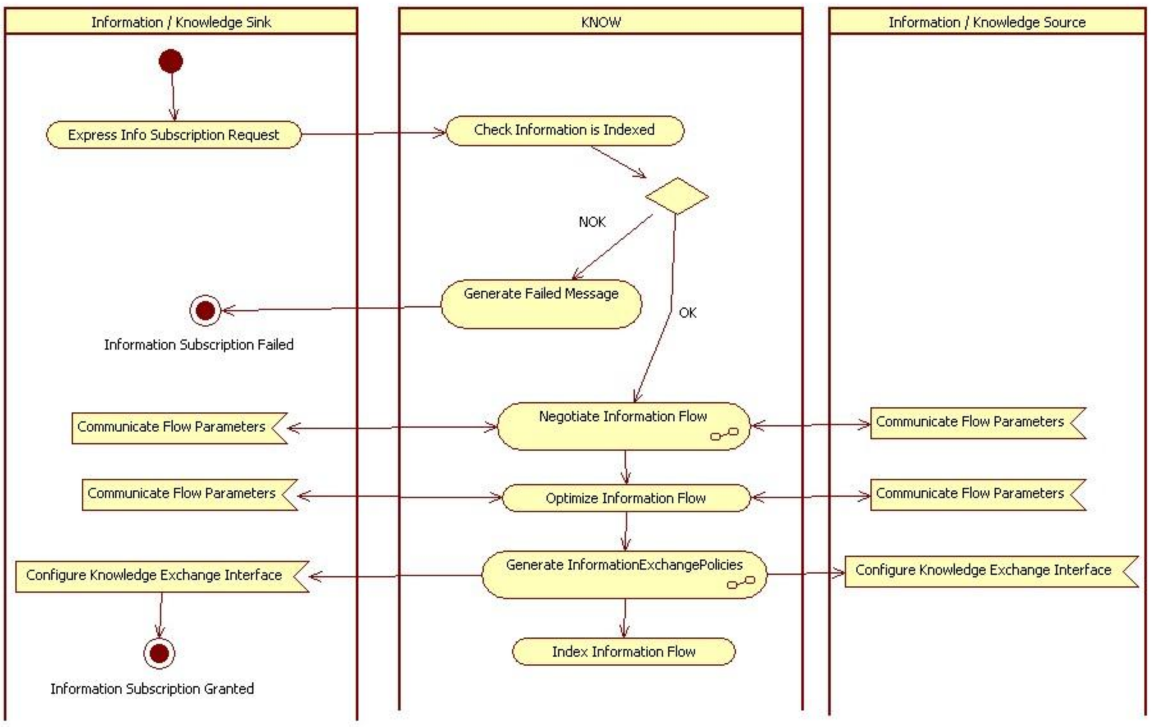

Information / Knowledge Sink
KNOW
Information / Knowledge Source
Express Info Subscription Request
Check Information is Indexed
NOK
OK
Generate Failed Message
Information Subscription Failed
Negotiate Information Flow
Communicate Flow Parameters
Communicate Flow Parameters
Optimize Information Flow
Communicate Flow Parameters
Communicate Flow Parameters
Generate InformationExchangePolicies
Configure Knowledge Exchange Interface
Configure Knowledge Exchange Interface
Index Information Flow
Information Subscription Granted

Figure 25. Information subscription workflow diagram (i.e., the Resolve Knowledge Dependencies process of the NEM registration diagram).

3.4 Coordination block

The role of the coordination block is to protect the network from instabilities and side effects due to the presence of many NEMs running in parallel. It ensures the proper triggering sequence of NEMs and their stable operation. To this end, the coordination block must define conditions/constraints under which NEMs will be invoked (i.e. produce their output), taking into account operator service and network requirements e.g. the needs to optimize the use of the available network resources and avoid conflicts between NEMs that can lead to sub-par performance and even unstable and oscillatory behaviours.

Alike any other UMF Core Block, COORD is also implementing at least a KnowledgeExchangeInterface in order to receive and provide flows of information under the control of the Information Flow Establishment and Optimisation function of KNOW (see section 3.3.4).

List of Coordination block functions

- Orchestration
- Optimization and Conflict Avoidance

3.4.1 Orchestration function

This function is responsible to address orchestration issues of NEMs. It relies on policies/scenario from the operator, corresponding input/output and timing relationships, to address issues such as ordering the execution sequence of NEMs and maintains the proper workflow in a way that is needed to resolve inter-NEM dependencies. An example of such operator policy would be “core segment optimization should follow access segment optimization”; in general policies dictated by the “service view” of the operator which can bind certain NEMs together, in order to ensure and maintain consistency in the service delivery.

Orchestration poses “lower-level” constraints that the running NEMs have to follow in order to avoid conflicts that could lead to under-performance.

List of “Orchestration function” operations

Update NEM instance description, Delete NEM instance description, Identify delta in COORD NEM registry

Name	**Update NEM instance description**
Description	Instance description of the registering NEM is stored in the COORD NEM registry. This operation must check if an older version of NEM instance description is already stored (and if yes, then it must update it).
Constraints	
List of input data	NEM Instance description (see section 3.1.6
List of output data	None
List of non-functional requirements	

Name	**Delete NEM instance description**
Description	Instance description of a NEM is deleted from the COORD NEM registry
Constraints	
List of input data	NEM Instance id
List of output data	None
List of non-functional requirements	

Name	**Identify delta in COORD NEM registry**
Description	This operation triggers the “Identify NEM Coordination needs” operation whenever a change in the COORD NEM registry is observed; either by the insertion of a new instance description (registering NEM) or by a change in the instance descriptions of the already registered NEMs
Constraints	
List of input data	NEM instance description of the registering NEM. Instance description of already registered NEMs (NEM registry)
List of output data	Trigger event for “Identify NEM Coordination needs”
List of non-functional requirements	

3.4.2 Optimization and Conflict avoidance function

This function is responsible for guiding the re-computation of the resource allocation to the NEMs in a way that optimizes the global system’s utility, capturing even the end-to-end optimization of different segments and for the detection and avoidance of conflicts between NEMs. Part of the role of this function is to group conflicts and assign feasible mechanisms to handle them taking into account:

- the available optimization and conflict avoidance mechanisms
- the dependencies between NEMs, as instructed by the Orchestration constraints

List of “Conflict avoidance function” operations

Identify NEM coordination needs, Choose coordination mechanism, Coordination mechanism feasibility check, Merge conflicts, Define coordination mechanisms parameters/NEMs control policy, Check coordination *mechanisms/NEMs control policy, Send NEMs Control Policy, Retrieve context/monitoring info, Retrieve NEM info, Call for Governance*

Name	**Identify NEM coordination needs**
Description	This operation consists in checking coordination needs between new conflicting elements and the managed ones. Based on this first assessment, this operation may update the list of conflicts between NEMs at an atomic level (e.g. From a NEM output to other NEMs using it).
Constraints	
List of input data	NEM instance description of the registering NEM. Instance descriptions of already registered NEMs (NEM registry) List of existing atomic conflicts
List of output data	Updated list of atomic conflicts
List of non-functional requirements	

Name	**Choose coordination mechanism**
Description	This operation consists in choosing coordination mechanism for each atomic conflict; the choice will be based on the available coordination mechanisms.
Constraints	
List of input data	List of atomic conflicts, NEMs instance descriptions including: -NEM ids to be handled by COORD, Parameters used (and where) -metrics affected (and where), -timings of the NEM (including both convergence time and expected interval between two triggers of the NEM) -Utilities Policies ORCH constraints Outcome of previous “coordination mechanism feasibility checks” Outcome of previous “Check coordination mechanisms/NEMs control policy” operations
List of output data	List of existing atomic conflicts associated with a coordination mechanism
List of non-functional requirements	

Name	**Coordination mechanism feasibility check**
Description	This operation consists in making a feasibility check for a coordination mechanism associated with an atomic conflict. This check is to verify and identify how far a

	coordination mechanism can manage an atomic conflict taking into account the NEM capabilities
Constraints	
List of input data	NEM instance descriptions, list of atomic conflicts ORCH constraints
List of output data	Feasibility grade, for the checked (atomic conflict, coordination mechanism) pair. If grade not satisfactory, to trigger Choose coordination mechanism OR Call for governance (the latter if no coordination mechanism is feasible for handling an atomic conflict).
List of non-functional requirements	

Name	**Merge conflicts**
Description	This operation consists in regrouping atomic conflict based on the coordination mechanisms
Constraints	
List of input data	List of atomic conflicts with associated coordination mechanisms ORCH constraints
List of output data	List of group of atomic conflicts with associated coordination mechanism
List of non-functional requirements	

Name	**Define coordination mechanisms parameters/NEMs control policy**
Description	Set parameters for the selected coordination mechanisms and produce the NEMs control policy for the NEMs that will need to be controlled by the selected coordination mechanism for each group of atomic conflicts
Constraints	
List of input data	Group of atomic conflicts with associated coordination mechanism NEM instance descriptions ORCH constraints
List of output data	NEMs control policy, Coordination mechanisms parameters
List of non-functional requirements	

Name	**Check coordination mechanisms/NEMs control policy**
Description	Check whether the coordination mechanisms when considered all together can operate as intended and whether the NEMs can enforce the instructions of the NEM control policy or there are some underlying restrictions that can prevent this in practice.
Constraints	

List of input data	NEMs control policies, coordination mechanisms parameters
List of output data	Ok/not ok; for the "not ok" case the error message should indicate which NEM control policy action(s) cannot be enforced and why, and/or which coordination mechanisms failed and why. If not ok, to also trigger Choose coordination mechanism OR Call for Governance (the latter when all possible coordination mechanisms have been checked for all possible groupings of atomic conflicts). This operation involves sending the candidate control to NEMs and receiving the outcome of its testing by NEMs.
List of non-functional requirements	

Name	**Retrieve context/monitoring info**
Description	Retrieve context and monitoring information from KNOW for use by the coordination mechanisms
Constraints	
List of input data	
List of output data	
List of non-functional requirements	

Name	**Retrieve NEM info**
Description	Retrieve NEM info from KNOW; information that may not be directly available from the COORD NEM registry but may be provided by KNOW (e.g. NEM convergence time, assuming KNOW logs and updates this for the NEMs of interest)
Constraints	
List of input data	
List of output data	
List of non-functional requirements	

Name	**Send NEMs control policy**
Description	Sends the NEM control policy to NEMs ; this includes disabling a NEM or configuring a NEM
Constraints	
List of input data	Outcome of "Check coordination mechanisms/NEMs control policy"
List of output data	Either ActionConstrainingPolicies or RegimePolicies (see Figure 6 page 20)
List of non-functional requirements	

Name	**Call for Governance**
Description	Alerts GOV when all the feasibility checks for a

	coordination mechanism to be associated with an atomic conflict have failed or when all coordination mechanisms have been checked and failed. Can also relay to GOV, Call for Governance messages received from NEMs.
Constraints	
List of input data	
List of output data	Call for GOV message (NEM ids, current output of "Check coordination mechanisms parameters/NEMs control policy", current output of "Coordination mechanism feasibility check")
List of non-functional requirements	TBD later

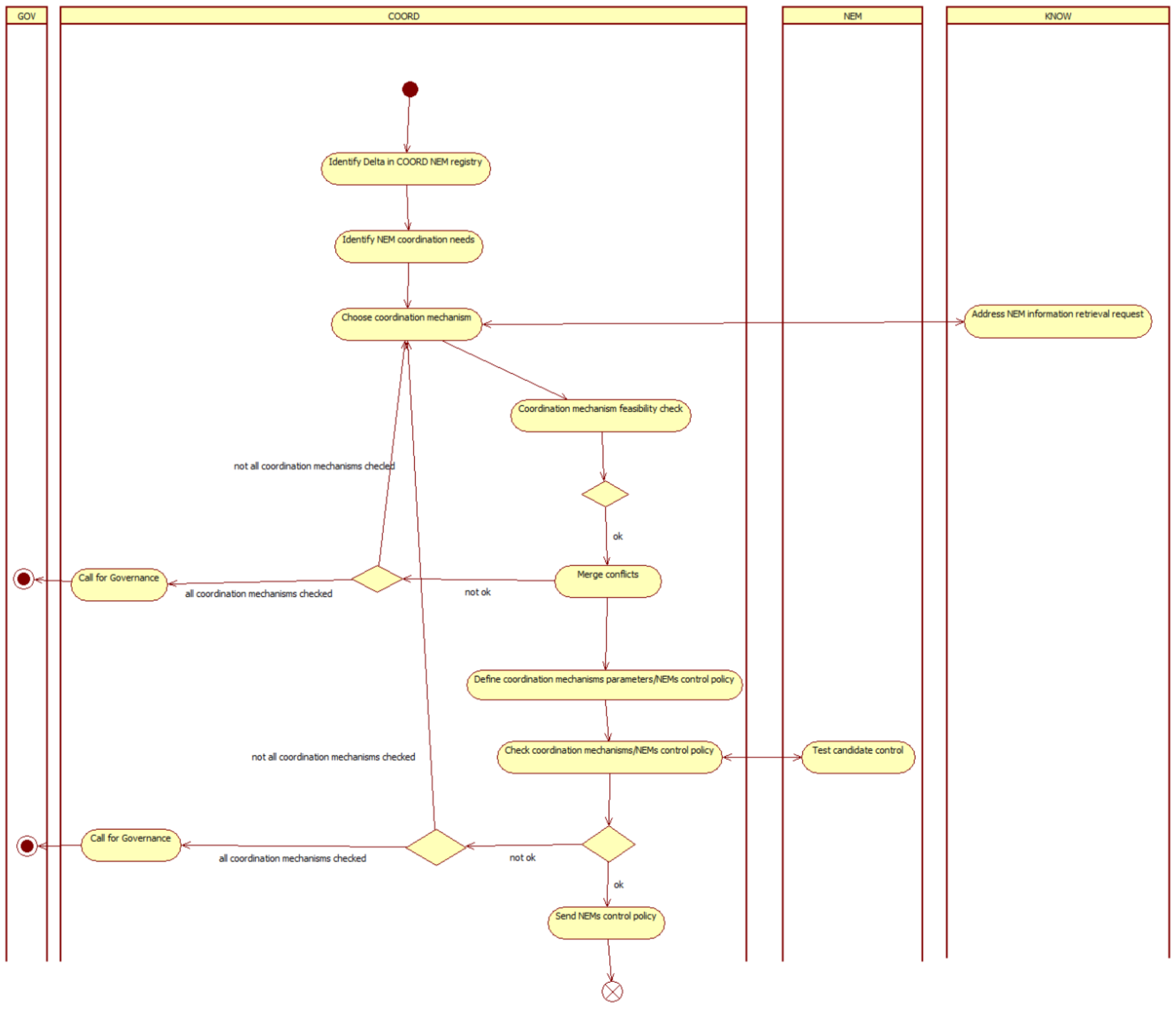

GOV
COORD
NEM
KNOW
Identify Delta in COORD NEM registry
Identify NEM coordination needs
Choose coordination mechanism
Address NEM information retrieval request
Coordination mechanism feasibility check
not all coordination mechanisms checked
ok
Call for Governance
all coordination mechanisms checked
not ok
Merge conflicts
Define coordination mechanisms parameters/NEMs control policy
not all coordination mechanisms checked
Check coordination mechanisms/NEMs control policy
Test candidate control
Call for Governance
all coordination mechanisms checked
not ok
ok
Send NEMs control policy

Figure 26. Manage conflicts activity diagram.

The workflow is triggered whenever a change in the COORD NEM registry is identified, whether by the insertion of a new NEM instance description or the updating of an existing one. This means that there may be a need to change the association of NEMs with the available coordination mechanisms or to update some parameters of the currently used mechanisms. Towards this end, first there is a re-assessment of the coordination needs between new conflicting elements and existing managed ones that leads to an updated list of atomic conflicts; then based on the availability of coordination mechanisms one of those is selected for each conflict. This may involve interactions with KNOW in order to retrieve possible NEM information that may be available in KNOW and help in the selection of the appropriate coordination mechanism.

This is followed by a check whether, taking into account the NEM capabilities, the selected mechanism can indeed be applied for managing the specific conflict. If the selected mechanism fails the check, then other coordination mechanisms are considered until a suitable one is found. If no suitable mechanism is found then GOV is notified of this situation. Assuming though that all atomic conflicts can be successfully associated with one of the available coordination mechanisms, then atomic conflicts are grouped and associated with a specific coordination mechanism (same instance of a coordination mechanism associated with multiple atomic conflicts) and the coordination mechanisms parameters and NEMs control policy are defined (hereafter NEM control policy can be either ActionConstrainingPolicy or RegimePolicy ore subscription request towards KNOW that will be concluded by KnowledgeExchangePolicy – see section 3.1.2 Figure 1. UMF overview and decomposition.).

This is followed by a check whether the coordination mechanisms when considered all together can operate as intended and whether the NEMs can enforce the instructions of the NEM control policy or there are some underlying restrictions that can prevent this in practice. If the checks are successful then the parameters are enforced in the coordination mechanisms and the corresponding NEMs control policies are sent to the NEMs. If however the check fails, then attempts are made to rearrange the association of atomic conflicts with coordination mechanisms and carry out again the grouping of them under specific instances of the coordination mechanisms. Eventually, if this process does not lead to a feasible solution then GOV is notified.

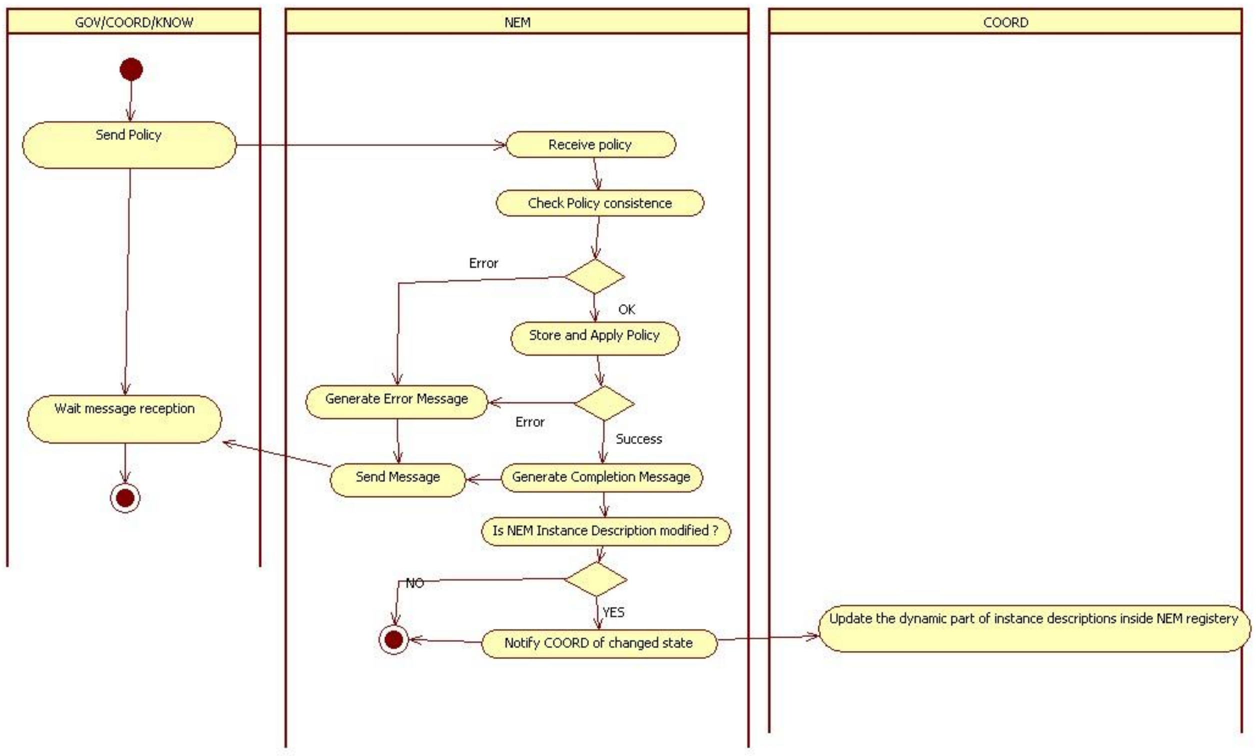


Figure 27. Set Policy activity diagram.

3.5 Interfaces

Interface Name	**GOV -NEM interface**
Description	Interfaces offered between NEMs and Governance. Governance uses it to retrieve information about the NEMs, to configure it through the mandate or policies, and to configure how the NEM is going to report information.
List of operations (exposed operations)	Send NEM Mandate Set Up a NEM Set Down a NEM Delete NEM Get NEM State Get NEM Mandate Revoke NEM Mandate Get Manifest

Interface Name	**GOV -COORD interface**
Description	The high level objectives of human operator are also transformed to a set of low-level policies for the cooperation of the NEMs (e.g. permissible cooperation of NEMs, in order to achieve a goal). Before this, coordination should inform Governance of the policies that can be customized and send to Coordination. Furthermore, Governance should inform the Coordination core component for the registered NEMs. So, the decision for NEMs' coordination would be taken in the framework that is defined from the policies of Governance and the requested information from Knowledge, corresponding to probabilities of specific operation/behaviour of registered NEMs in particular network conditions. Finally, the possibility of COORD calling to governance when the available coordination mechanisms cannot successfully achieve their goals is also included here.
List of operations (exposed operations)	Inform Coord Policies Set Coord Policies Call For Governance

Interface Name	**Human operator - GOV interface**
Description	Interface that allows the interaction of the human network operation and the UMF. This interface allows: the definition of Services; The high level objectives of human operator are also transformed to a set of low-level policies for the cooperation of the NEMs (e.g. permissible cooperation of NEMs, in order to achieve a goal);

	and the visualization of the network status
List of operations (exposed operations)	Define Service Define High Level Objectives Define Coordination Policies Visualize Network Status Send Notification

Interface Name	**Knowledge Exchange Interface**
Description	This interface is part of KNOW block and is responsible for exposing KNOW operations which are related to information exchange to the other UMF core blocks, i.e. GOV and COORD blocks, and to the NEMs. For example, from the COORD point of view this interface can be used as follows: a) for retrieving the aggregated information on conflicts between NEMs which is generated by information aggregation and not explicitly by the NEMs. b) for knowledge provision based on which, joint optimisation and orchestration of the NEMs can be performed c) for COORD to provide the status of its operations to KNOW
List of operations (exposed operations)	Information Collection Information Sharing Information Retrieval Information Dissemination

Interface Name	**Knowledge Management Interface**
Description	This interface is part of KNOW block and exposes to the other UMF core blocks and the NEMs those KNOW operations that are related to management issues of KNOW. For example, policies, aggregation mechanisms to be used, optimisation goals for the KNOW, and the configuration of the KNOW properties (e.g., to change the information flow optimisation policies, to add new general accuracy objectives for the information filtering etc) can be passed to KNOW via this interface.
List of operations (exposed operations)	NEM registration Information quality controller

Interface Name	**COORD -NEM interface**
Description	Interfaces offered between NEMs and Coordination. Coordination uses it to request information from NEMs that is needed by the coordination mechanism and test/enforce changes in the NEM behavior. NEMs

	use these interfaces to respond to this testing/enforcement of changes in their behavior and also request a control policy when needed.
List of operations (exposed operations)	Send NEMs control policy Request/ Respond NEM control policy Call for Governance Send candidate control Reply candidate control Unregister NEM

Interface Name	**COORD -KNOW interface**
Description	Interfaces offered between Knowledge and Coordination. Coordination uses it to request information from Knowledge that is needed by the coordination mechanisms. This information can be both network/context related but also NEM related
List of operations (exposed operations)	Request NEM information Send NEM information Request context/network information Send context/network information

4 UMF core mechanisms

4.1 Governance mechanisms/tools

4.1.1 Translation mechanisms

The translation of business goals to low level policies (well known as "policy refinement" as well), which encompass semantics and reasoning techniques, accomplish the successful conversion of higher level policies to lower level policies, enabling policy continuum and business goals realization.

The stages of policy translation realization are:

a. Specification of the kind of information that is conveyed by the corresponding policies. The respective information flow of "Business level" policies is related to business-level goals and service requests, the information flow of "Service level" policies is related to service characteristics as these reflect to specific network parameters, and the respective info flow of "NEM" policies is related to demanded operation/behaviour/usage of NEMs/elements/resources in specific network segments.
b. Definition of relevant information classes with corresponding semantics for each level.
c. Determination of the relation between classes of adjacent levels.
d. Design of the alternative form of policies per level with the assistance of proper operators, based on the corresponding operations/processes and NEMs coordination, and the correlation with the defined classes.

In essence, the policy manager of each level receives the policy, recodes it and proceeds to its deconstruction to a set of instances of info classes. In case, that there are not any relevant classes, policy managers are "learning" to create new classes and associate them with classes of the upper and lower level.

The design of the policies structure may be effectuated in the framework of the existent policy model that is related to the selected information model (SID).

For example in UC6 demo, the administrator writes to the console of the H2N tool: "I want to serve new traffic consisting of 300 mobile concurrent users of Video Conference Application with SLA corresponding to high QoS, on top of my multi-vendor and multi-technology infrastructure in a reliable manner, for the centre of Piraeus, between 4.00pm and 4.30pm". This phrase is transformed to a Business level policy entitled "BusinessLevelEntryNotification". Specifically, the info elements are identified and the respective values are registered to the corresponding info classes. So, the instances of the classes: OperatingScheme, NumberOfUsers, GovLocationInfo, GovTimezoneInfo, GovApplicationInfo, are produced. Some of the classes may consist of an array of other classes, as for example the GovApplicationInfo. In this case, the instances of the classes are shown in Table 8.

Table 8. Instances of classes of "BusinessLevelEntryNotification" policy

Business level policy classes		Instances
OperatingScheme		reliable accommodation of traffic of all users
NumberOfUsers		300
GovLocationInfo		centre of Piraeus
GovTimezoneInfo		between 4.00pm and 4.30pm
GovApplicationInfo	GovApplicationInfo	Video Conference Application
	GovQualityLevel	High QoS

These classes are combined (based on the affiliated semantics, the structure of policies in the framework of the selected policy model and the syntax rules of the selected policy language), and construct the business policy "BusinessLevelEntryNotification".

This business level policy is sent to the suitable entity (entity that realizes the relevant SAD_FB functionality in this case), which analyses the received policy and generates the instances of the classes that are associated with the contained info elements. Specifically, she generates the instances of the classes NumberOfUsers, GovLocationInfo, GovTimezoneInfo, and targets at generating instances of the classes that are associated with GovApplicationInfo [Application, GovQualityLevel]. In order to achieve this, she obtains updates of the rules for association of services to users classes and quality levels from the respective Repository, corresponding to specific Quality of Service parameters. These rules constitute another business level policy entitled "AssociationNotification" policy. The contained info of the policy is assigned to classes: UserClassesOfApplication, QualityLevelParametersOfApplicationOfUserClasses. Finally, based on this policy set, the policy manager of the service level generates instances of the classes: NumberOfUsers, GovLocationInfo, GovTimezoneInfo, Application, QualityLevelParametersOfApplication.

These classes are combined to make up the corresponding service level policy that is sent to the proper entities (which manage the potential targeted NEMs). After the suitable operations of the responsible entities, which are triggered by this service level policy, the base stations in the determined area are identified, and based on the estimated load (probability) for each of them, the base stations (RAN elements) that will be used are selected and the relevant calculated values are assigned to the corresponding classes. Then, the policy manager of the entity that effectuates the CSC_FB functionality, combines the relevant instances of classes Application, NumberOfUsers, RANelementParameters, with the proper operators, formulates the resource policy "NetworkOfferRequest" for each selected RAN element (NEM policy), and sends it to them.

The presented approach enables the selection of different implementation policy languages, according to the needs and the particular goals of the operator. A high level representation of the policy continuum in this instantiation of UC6 is depicted in the Figure 28.

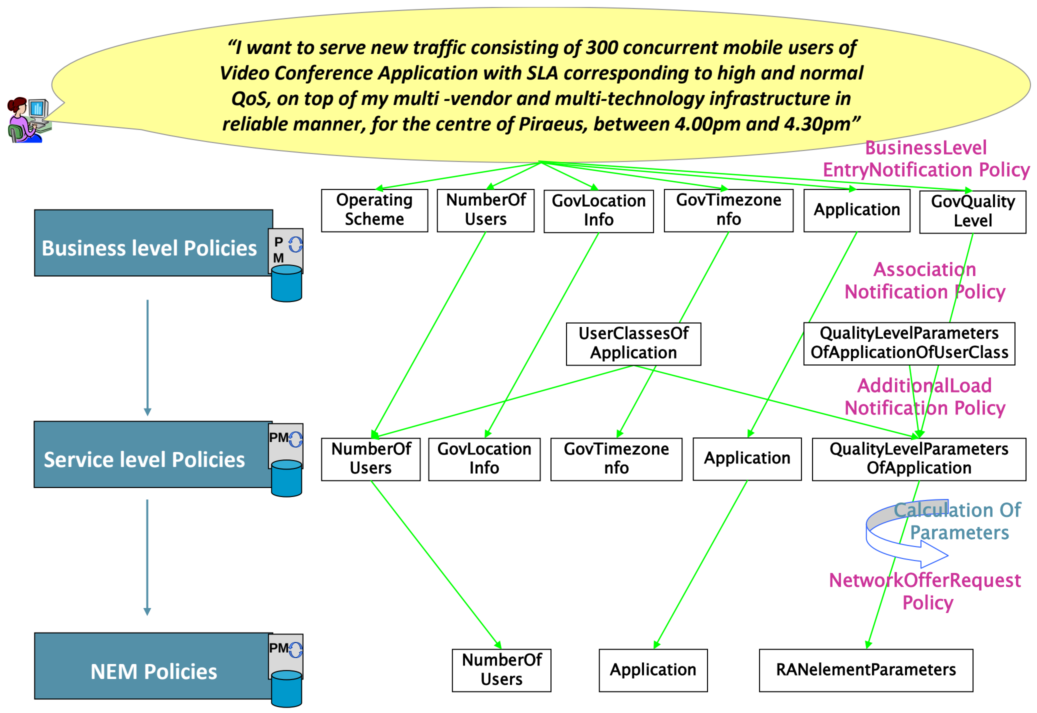

"I want to serve new traffic consisting of 300 concurrent mobile users of Video Conference Application with SLA corresponding to high and normal QoS, on top of my multi -vendor and multi-technology infrastructure in reliable manner, for the centre of Piraeus, between 4.00pm and 4.30pm"
BusinessLevel EntryNotification Policy
Business level Policies
PM
Operating Scheme
NumberOf Users
GovLocation Info
GovTimezone nfo
Application
GovQuality Level
Association Notification Policy
UserClassesOf Application
QualityLevelParameters OfApplicationOfUserClass
AdditionalLoad Notification Policy
Service level Policies
NumberOf Users
GovLocation Info
GovTimezone nfo
Application
QualityLevelParameters OfApplication
Calculation Of Parameters
NetworkOfferRequest Policy
NEM Policies
NumberOf Users
Application
RANelementParameters

Figure 28. High level representation of the policy continuum for the instantiation of UC6.

It is noted that the policies of this instantiation of UC6 have a "Request" form, which has not the classic form of Event-Condition-Action policy of Policy SID model. For example, for the representation of the "BusinessLevelEntryNotification"policy in the context of Policy SID model, although it can be considered that the PolicyActions are of the form "SET <action-target> to <value>", there is not PolicyCondition of the form: "IF <policy-condition> is TRUE", and the definition of PolicyConditionSpec complicates attempts unnecessarily to adapt the high-level governance policies and associations to SID framework. This fact emphasises the need for extension of the SID policy model, in order to cover all the possible forms and cases that will accrue from different requirements in the future networks and systems. Furthermore, in the framework of extension of SID policy model, it would be helpful to study extensions of a set of defined attributes, for example in the context of PolicyRuleBase and PolicyRuleSpecifications (e.g. the alternatives choices for executionStrategy).

Based on the above approach for policy translation, future work may comprise:

- Mapping of parameters of tentative messages of Use Cases in D2.1 (with probably necessary updated information) based on the selected IM (SID) (Cooperation of WI2 and WI6).
- Delegation of info classes per policy level and definition of proper semantics
- Definition of the policy model- possibly extension of SID policy model
- Selection of policy language per level.

For the realisation of the translation (refinement) of the high level policies (resides in highest level of policy continuum) to low level policies (resides in the lowest level of policy continuum), a variety of policy refinement methods have been proposed so far with certain pros and cons, the most typical of which are:

- Goal-oriented policy refinement
- Classification-based policy refinement
- Ontology-based policy refinement
- Prescription-based policy refinement
- Case-based policy refinement

In general, a key challenge in developing a policy translation approach is to achieve an acceptable trade-off between the generality of the approach and the level of automation possible. Fully-automated approaches are also highly specialised to particular applications domains and cannot be applied to other domains. On the other hand, generalised approaches to refinement require experts who are familiar with both the application domain and low-level formal representations to provide information regarding the managed system.

After studying in detail the aforementioned approaches, the most efficient and wide accepted seems to be the ontology-based policy refinement methods. The use of ontologies to represent each level of the policy continuum, and the definition of mechanisms for translation using semantic techniques, made this approach valid with independence of the number of policy continuum levels selected to be implemented (e.g. 5 or 3 levels). In this direction, we can assume that each vendor will have its own ontology for the lowest level, while an ontology for the business layer will be defined, assuming the n intermediate levels have also their own ontology. The translation methodology will describe how the translation can be done (mapping between ontologies); probably imposing requirements on the policy model (see [2][1]). Then, at implementation time, one has to select the most appropriate number of levels for his particular problem.

Among the various ontology-based policy refinement methods the most prominent candidate is the use of OWL/SWRL for the representation, translation and reasoning of policies, for the following reasons:

- It is a general approach which can be applied to a variety of application and technological domains in contrast to other methods which require high human intervention (e.g. prescription-based policy refinement, case-based policy refinement)
- It is a highly automatic approach. The generation of ontologies and SWRL rules ensures the automaticity of the translation process.
- Support interoperability between high level policies and low level policies, enabling bidirectional information mapping at runtime. This approach is the only one (to the best of our knowledge) that supports bidirectional refinement of policies.
- It is not as complicated as other methods (e.g. goal-oriented policy refinement)
- Provide interoperability with other information models (apart from SID), policy models, and behaviour definitions since they are represented in OWL and SWRL.
- Can be interpreted and executed by general purpose semantic engines.
- Can be implement with reasonable resources.

Another approach of realising the policy refinement procedure and network governance framework in general, considers KPIs as of crucial importance. More specifically, the performance of the operations of a telecommunication system as well as the performance of the services provided can be described by a set of KPIs. The current values of KPIs designate the current efficiency of the network in accomplishing the operational targets, while also indicating the overall performance of the services in fulfilling a certain level of quality defined in users' SLAs (Service Level Agreements). In this direction, the proposed methodology defines the KPIs as the cornerstone of policy translation process. In practice, all the available KPIs do not equally affect any possible business goal. On the contrary, specific types of goals are affected by sets of specific KPIs. Therefore, in order to constrain the policy translation process to only meaningful translations, we classify the available business goals into categories based on areas of interest and assign to each category a set of KPIs (e.g. category “QoE” includes the following KPIs: Delay, Jitter, Packet Loss).

GoalCategoryi = {KPI1, KPI2, …….KPIn}

The selected KPIs prove to affect (based on standards) the goals of the specific category in terms of network and service performance. In addition, the aforementioned KPIs can be either operational or service based KPIs (e.g. KPIs described in SLAs) reflecting the ability of the translation framework to co-manage network and services.

In the meantime, the majority of policy translation studies concentrate on refining an initial goal into a set of low level policies that are generated dynamically from scratch. This approach, although general enough, lacks practical feasibility as it requires the operator to provide a vast quantity of information that is very difficult or even impossible to collect. In addition, the derived policies should be verified by the operator prior to their enforcement as they may be unexpected and may drive the network to instability.

The proposed methodology is based on the reusability of policy templates for the generation of policies in all the layers of policy continuum. An indicative example of a policy template is depicted in Figure 29. According to this approach for each Goal Category or KPI set, a series of policy templates (based on OWL) are already stored in a pool of policy templates. In practice, these templates form the policy skeleton on which the real policies will be built during the translation phase. In the pool, some policies are linked reflecting the translation step, while others are subclasses of more general policy templates. These templates are generated by network experts and after their creation they can be reused or extended to meet the needs of the specific operator.

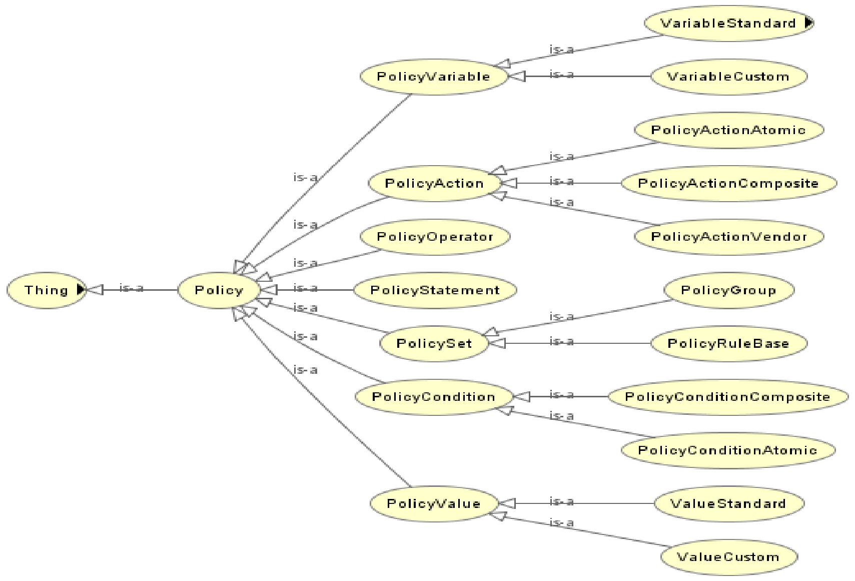


Figure 29. Policy template in OWL.

A vital input to the translation process is the network hierarchy and topology. This information is necessary to map the abstract entities (described in OWL) to real network devices, objects and operations and subsequently, to enforce the derived low level policies to selected network components. Additionally, in the proposed approach this information is used for the translation of policies between two subsequent levels of policy continuum. For example, the managed network may logically consist of a number of heterogeneous domains (e.g. LTE, WiFi, IP Core), each consisting of a number of network elements (e.g. access points, eNodeBs, routers, links), which also consist of the relevant supported operations and manageable objects. The translation process will refine the initial business goal to a number of sets of policies, reflecting this hierarchy (e.g. policy sets for LTE, WiFi, IP Core domains and subsequent policy sets for access points, eNodeBs and routers). This network hierarchy related information can be extracted from the network inventory system or can be derived by using a number of available tools that automatically generate network topology data using discovery techniques.

In line with the above, the proposed policy translation methodology relies on ontology-based policy refinement approaches and it is in line with the translation methodologies studied in [3] and [4]. The translation process, which is described in detail below, makes use of OWL/SWRL for the representation, translation and reasoning of policies. In addition, the translation process adopts the Policy Continuum approach presented in [5]. In the example illustrated in Figure 30 of five different levels / views are defined, each of which constitutes a different representation of the initial business goal.

In each level of the policy continuum, a series of OWL classes is defined, enriched with object and data properties in order to express semantic relations. These classes conform to the considered Policy Ontology, by means of inheritance. Traversing the Policy Continuum in a bottom-up approach, OWL classes represent network concepts in a natural-language oriented approach. For instance, the concept of "USER" is defined in each level of policy continuum, as it can be depicted Figure 30 with different data properties in each layer (i.e. User Class, hasUC).

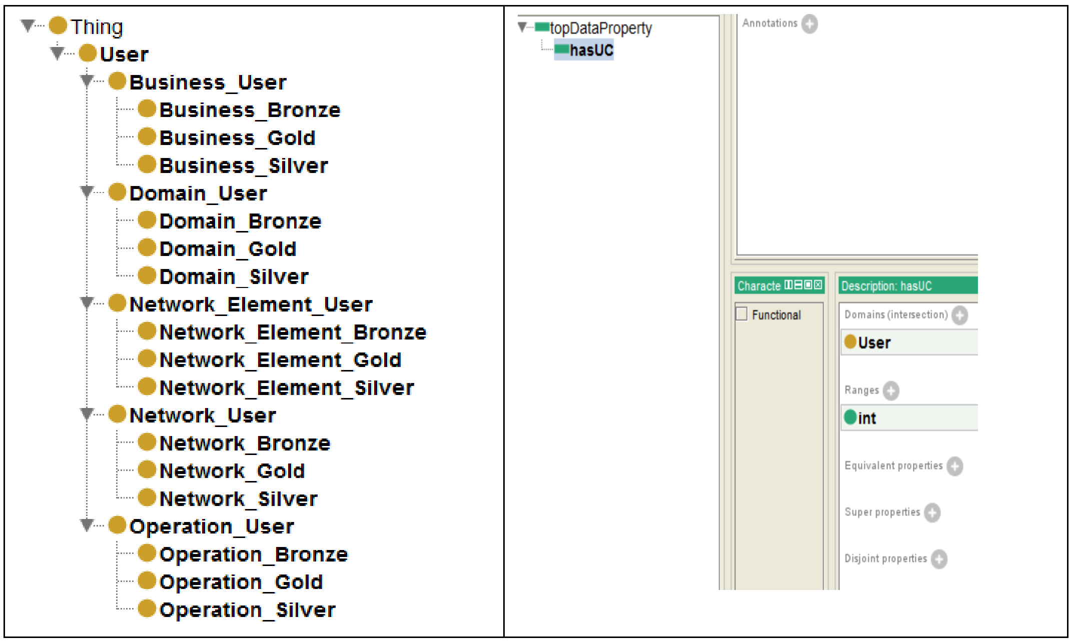


Figure 30. The concept of USER in OWL.

The transition between consecutive policy continuum levels is achieved through the use of SWRL rules. SWRL is a W3C [6] specification that combines OWL DL and RuleML languages. Rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Considering the examined approach, SWRL rules constitute the intermediate among policy continuum levels and achieve the translation from business to operation level concepts. An indicative example of SWRL rule is as follows,

Policy(?p) AND hasBussinessUser(?p, ?buser) AND Business_Gold(?buser) AND hasNetworkUser(?p, ?nuser) -> Network_Gold(?nuser) AND hasUC(?nuser, 1)

For the considered SWRL rule, the antecedent part examines if individuals of Business_Gold class exists. If so, a one-hop step to the Network level exists where the user individual is assigned a semantic property value. As a consequence, the definition of both OWL classes and SWRL rules provide us with ability of simplified modelling and automate reasoning. No extra effort is considered, apart from the initial specification (on business level) of the appropriate OWL-class individuals.

The translation process (for the case of 5 policy levels) is illustrated in Figure 31. The translation process comprises the following steps (for simplicity we assume 5 level of policies, while this approach can be easily configured to support any number of policies selected):

Step 1: The initial business goal (relying on the business goal level of policy continuum) is classified to a Goal Category based on operator's selections

Step 2: The business goal is translated to one or more network level policies based on a) Goal Category to KPIs mapping and b) combination of high level semantics (e.g. "High level of Speed") and service classification (e.g. Transactional service) to KPI values (e.g. $KPI_{Delay} < 50$ msec) mapping.

Step 3: For each network level policy a series of template policies (for all the subsequent levels) are extracted from the policy template pool. The selection of the appropriate policy templates is done based on the set of KPIs involved and the initial goal classification. In general each policy template is a policy skeleton that contains the policy structure and the policy variables, while the values of variables are missing. During the translation process the missing values are filled and the real policies are generated.

Each network level policy is translated to a set of domain level policies. This translation is based on: a) actual network hierarchy (the composition of network to domains and the type of each domain), b) the translation algorithm and c) the domain level template policies.

Step 4: The translation algorithm specifies the way in which a KPI performance or a parameter value is shared / split among the available domains. In its simplest form the algorithm translates KPI / parameter values proportionally among domains based on weights assigned to each pair of {domain, KPI/parameter}. An example of domain weights per KPI/parameter is illustrated in Table 9. Thus:

$$KPI_{d,i}^{DomLevel} = W_{d,i} * KPI_{i}^{NetLevel}$$

where $KPI_{d,i}^{DomLevel}$ is the value of KPI i for the domain d on domain level, $KPI_{d,i}^{NetLevel}$ is the value of KPI i on network level, and $W_{d,i}$ is the weight assigned to domain d and KPI i.

More complex algorithms can be implemented as well, or different algorithms can be used based on the number of policy levels and the algorithm's objectives without losing the generality of the proposed approach. The weight values can be specified by the operator or can be estimated by the framework by using knowledge building functionalities (e.g. initially all weights are equal, while during network operation weights are modified automatically by knowledge components based on monitoring) and therefore, increase the level of automation (minimum operator effort).

Then, the domain level policies are generated based on the domain level policy templates selected in step (3). During this phase, the generated policies are customised in terms of filling / extending the policy skeletons according to the above estimations of KPI and parameter values.

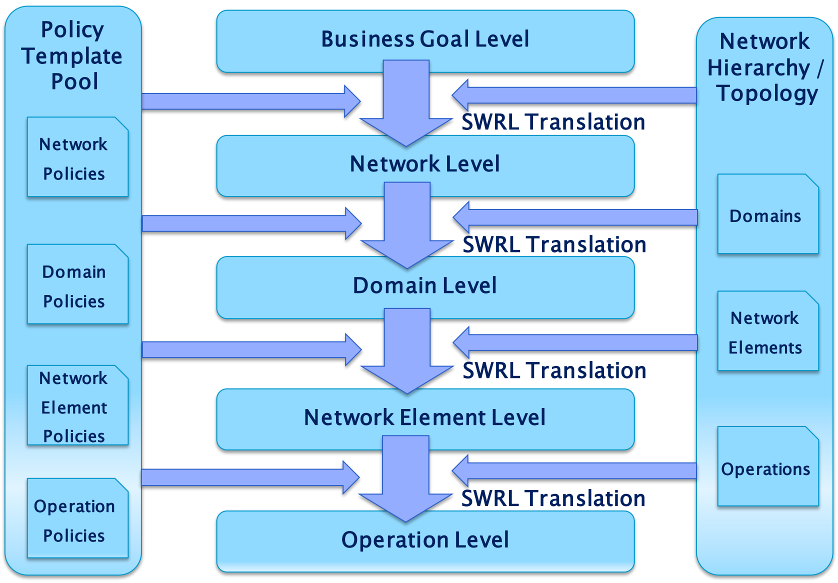


Figure 31. Policy translation process.

The SWRL rules used support KPI evaluation, by adding class relations and swrl semantics (e.g. swrl:greaterThan(), swrl:equal()).

Step 5: Each domain level policy is further translated into a set of network element level policies. This process is similar to step (4). The translation is based on: a) actual domain hierarchy (the composition of domain to network element), b) the translation algorithm and c) the network element policy templates. The translation algorithm uses weights in order to translate KPI/parameter values on domain level to respective values on network element level. Then, policies are generated based on the network element level policy templates selected in step (3), properly extended to include the results of the translation algorithm.

Step 6: On the final level each network element level policy is further translated to a set of operational low level policies. The translation is realised by generating operation policies from the operation policy templates and enhancing them with KPI/parameter related information. In this step a mapping is realised between the involved KPIs/parameters and the available operations/functions of the network elements. In practice, the derived operations are already described in policy templates, while the operation parameter values are determined by the translation algorithm.

Table 9. Domain Weights per KPI/Parameter

Domain i							
KPI weights				**Parameter weights**			
KPI1	KPI1	....	KPIn	Par1	Par2	....	Parm
$W_{KPI,1}$	$W_{KPI,2}$	....	$W_{KPI,n}$	$W_{par,1}$	$W_{par,2}$	....	$W_{par,m}$

4.1.2 Policy validation, conflict detection and resolution

Policy conflict detection should be performed in all stages of policy translation between the different policy levels. If the outcome of the policy conflict detection is that the conflict cannot be resolved, a proper mechanism has to be designed, which will translate the conditions that led to this decision to a human friendly formulation, for the enlightenment of administrators/operators. Based on this, it may be required the alteration of specific business goals from the operators.

In this section a brief description of policy conflict validation and detection strategies is provided. Firstly, we present the state of the art approaches that are available in the literature and have been used as a basis for the production of our approach.

Policy validation, policy conflict detection and conflict resolution have attracted research interest over the past years. In [7] the authors describe a set of methods that are necessary to be applied in order to resolve conflicts. More specifically, approaches for monitoring conflicts at run-time and for conflict resolution are analyzed. Authors' approach is based on their previous work on the specification of policy model [8] and the methods they developed for conflict detection [8][9][10][11].The approach in [7] has set the basis for [12] where the authors present an approach towards policy conflict analysis based on the formalization and reasoning provided by Event Calculus and its application in the domain of DiffServ QoS management. Finally, in [13] an approach that is based on free variable tableaux for detecting conflicts resulting from the combination of various kinds of authorization and constraint policies used in Web Services environments is introduced. The method not only enables static detection of policy conflicts such as modality and static constraint conflicts (i.e. propagation, Chinese wall, time constraint etc.) but also yields information that is helpful for correcting the policies. The approach can be applied to various policies written in different policy definition languages.

In the methodology followed, a transformation of the problem of anticipating conflicts between policies into an ontology consistency checking problem will be studied. This process could be based on this conflict resolution cycle: identify-classify-detect-resolve. Future work for conflict resolution comprises the investigation of static and dynamic conflict detection strategies as well as studying conflict resolution strategy based on the establishment of policy precedence as it also suggested in TM Forum.

In the followed approach policy conflict detection is performed in the final stage of policy translation (i.e. operation level). In case the outcome of the policy conflict detection is that the conflict cannot be resolved, a proper mechanism has to be designed, which will translate the conditions that led to this decision to a human friendly formulation, for enlightenment of administrators/operators. Additionally, a potential the requirement that should also be taken into consideration is the requirement for alteration of specific business goals from the operators, in order to allow as many policy rules as possible.

Policy Conflict Resolution (PCR) module in our case is considered as part of the Policy Decision Manager logical block. As shown in Figure 32 PCR interacts with the Policy Translation module. More specifically, PCR receives as input the outcome of the Policy Translation procedure. This outcome is formed as a PolicySet object that conforms to the TM Forum standards. This object includes all the low level policy rules (i.e. policy rules that can be understood by the NEMs) that should be applied to the NEMs. The result of the PCR is the production of a conflict-free Policy Set object. This outcome is passed then to the Distributor and so on until they are applied to the NEMs. Future plans of our work include the production of a reverse process regarding policy translation, which will produce the refined Business Goals based on the new Policy Set object produced by the PCR module and will provide these Business Goals to the operator.

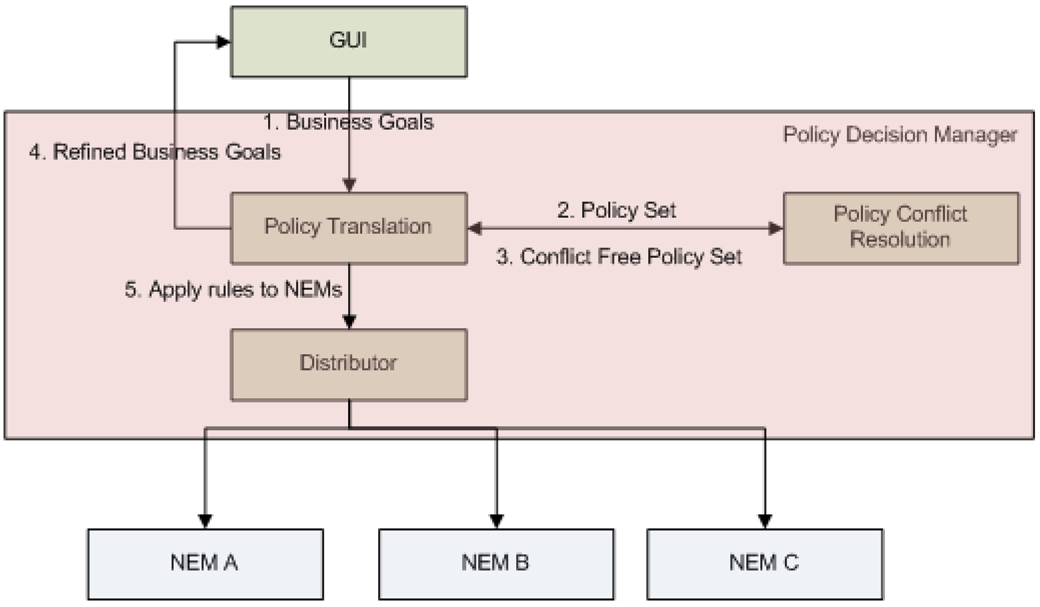


Figure 32. Policy Conflict Resolution interaction.

Figure 33 presents the steps of execution of the Policy Conflict Resolution module. As mentioned afore, PCR receives as input a Policy Set object from the Policy Translation module. Then PCR is executed in 3 logical steps:

- Identification of newly expressed rules and already deployed rules
- Conflict detection and suggested actions for resolution
- Production of conflict free policy set

At the first step PCR receives the Policy Set object and decomposes the list of Policy Rules included in it to 2 lists (i.e. one with the deployed policy rules and one with the new policy rules) based on a cache memory preserved. Then, the second step is further split in 3 steps checking for conflicts regarding the events, the conditions and the actions respectively. It should also be noted that during the conflict detection on conditions of the policy actions a set of conditions that do not lead to conflict and could replace the initial policy conditions is produced. Furthermore, when the algorithm is checking for conflicts on actions a complementary predefined matrix, namely Service Interdependencies matrix is checked to evaluate whether two actions are conflicting or not. Finally, a conflict-free policy set object is produced and returned to the Policy Translation module. This policy set contains the modified Policy Rules that do not contain conflicts. Also, the Cache memory with the deployed and new rules is updated.

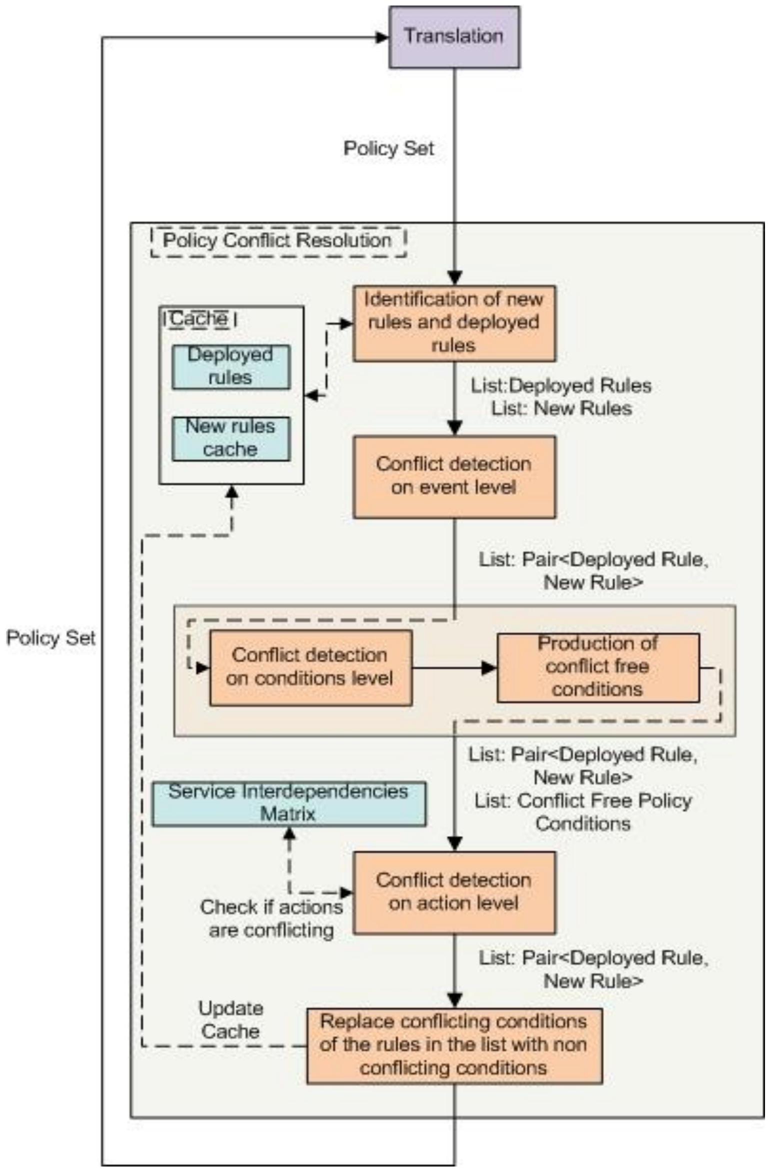

Translation
Policy Set
Policy Conflict Resolution
Identification of new rules and deployed rules
Cache
Deployed rules
New rules cache
List:Deployed Rules
List: New Rules
Conflict detection on event level
List: Pair<Deployed Rule, New Rule>
Policy Set
Conflict detection on conditions level
Production of conflict free conditions
List: Pair<Deployed Rule, New Rule>
List: Conflict Free Policy Conditions
Service Interdependencies Matrix
Check if actions are conflicting
Conflict detection on action level
List: Pair<Deployed Rule, New Rule>
Update Cache
Replace conflicting conditions of the rules in the list with non conflicting conditions

Figure 33. Policy Conflict Resolution flow.

4.1.3 Policy assessment mechanisms

The successful translation of high level to low level policies is of high importance from the operator's point of view. Term "successful" is used to express the sufficiency of the derived policies to accomplish the goals described by the operator in the high level policies. A successful policy will lead to well controlled and efficient network operations, while an unsuccessful policy may lead to misconfigurations, QoS / QoE degradation and network instabilities. Thus, a mechanism able to evaluate the policy translation process and measure the gains from the policy application is necessary. The success of a policy in accomplishing the goals described by the operator is in strong relation with the trustworthiness of this specific policy. Trust of policy can be defined as a comparison between the reference behaviour (the behaviour implied in high level policies) and the actual behaviour (based on measurements) of the network after the implementation of the policy. According to this, policy trustworthiness is a measure of policy assessment.

In order to estimate Trust, the Entropy-based trust model [15] can be used which uses uncertainty as measure of trust. In the proposed method of trust estimation, the concept of trust describes the certainty of whether the implemented policy will fulfil the objectives described in the high level goals. Information theory states that entropy is a nature measure for uncertainty [16]. Thus, entropy-based trust value is defined as:

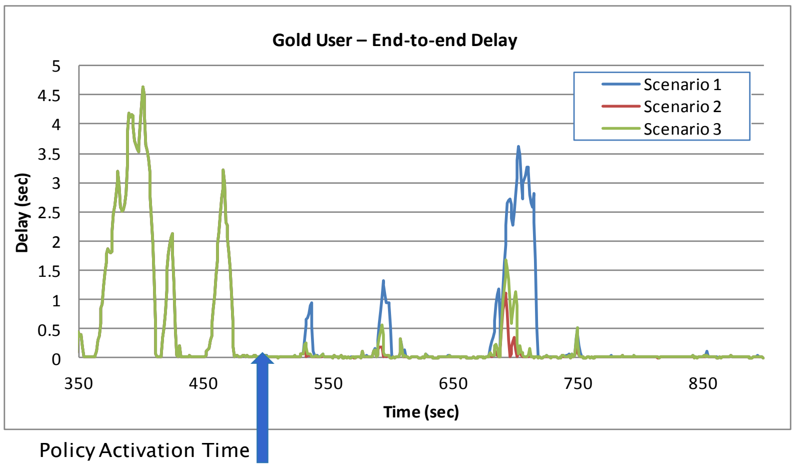
$$T\{subject : policy, successful\} = \begin{cases} 1-H(p) & 0.5 \le 0 < 1 \\ H(p)-1 & 0 \le p < 0.5 \end{cases}$$

where H(p) = −p log2(p) − (1 − p) log2(1 − p) is the entropy function and p=P{policy,successful}.

According to this formula, the trust value is a continuous real number in [-1, 1]. This definition satisfies the following properties. When p=1, the subject trusts the policy the most and the trust value is 1. When p=0, the subject distrusts the policy the most and the trust value is -1. When p=0.5, the subject has no trust in the policy and the trust value is 0. In general, trust value is negative for $0 \le p < 0.5$ and positive for $0.5 \le 0 < 1$. Trust value is an increasing function of p.

According to the aforementioned trust methodology, the policy assessment function calculates p after the implementation of a policy, based on network measurements collected by agents (reflecting a set of KQIs and KPIs according to TM Forum SLA Management Handbook) and assigns to this specific policy a value of trust. The estimated values of trust for specific policies can also be presented to the operator through the H2N Governance GUI in order to be able to supervise and control the underlying autonomic functioning.

An example of Trust of policy estimation is illustrated in Figure 34. Trust of policiesIn this scenario the trustworthiness of three candidate policies are estimated based on network measurements. In detail the following KPI is selected for policy evaluation: 90% of the end-to-end packet delay values should be below 200msec. The value of P is calculated after the implementation of the policy, while it is assumed that a policy is successful if the KPI is valid (Trust value > 0). The Table the estimated values of p, H(p) and policy trustworthiness according to the aforementioned methodology. The assessment of policy translation indicates that both policies are successful. In addition, as far as Trust of policy is concerned, it becomes obvious that policy of scenario 2 is more trustworthy than policy of scenario 3. In reality, policy of scenario 3 is untrustworthy at all as its value is near 0. In a real implementation, assuming that the Trust threshold for policy evaluation process is set to 0.3, the policy of simulation scenario 3 will be rejected, having policy of scenario 2 as the only candidate solution.



Policy	p	H(p)	Trust
Scenario 1: Absence of high level policies	-	-	-
Scenario 2: A high level policy is translated to specific low level actions	0.8341	0.648	0.3519
Scenario 3: A high level policy is translated to different low level actions	0.6708	0.914	0.086

Figure 34. Trust of policies

4.1.4 Network supervision mechanisms

One of the most important decisions is about which information to show so to fulfil the requirements from the human network operators. The main QoS factors contributing to service performance are service support, operability, accessibility, retainability, integrity, security and performance [14]. Therefore, each service performance can be characterised by a combined set of the aforementioned QoS factors. The parameters chosen as contributors to the QoS factors may be service specific, technology specific, or service and technology independent parameters. The parameters selected are those that are fundamental to the service and affect the customer's experience. The selected parameters which are reflecting the selected KQIs (Key Quality Indicators) of the services and the relative KPIs (Key Performance Indicators) can be monitored in real time by the operator thought the H2N interface. In addition, alarms can be programmed (manually or automatically) to be generated in case of performance degradations or violations (e.g. SLA violations).

The chosen parameters can be further combined into a single QoS value or index (to be presented to the operator) which represents the delivered QoS and provides an overall view of how well the delivered service meets the committed service.

4.2 Information and knowledge management mechanisms

4.2.1 Information collection and dissemination mechanisms

We discuss below a number of ICD mechanisms or algorithms that could be potentially used in the corresponding sub-block. At this stage of work, we selected mechanisms that partners of the consortium are already familiar. Of course, this list of mechanisms is not exhausting, since it is an ongoing work to evaluate or suggest appropriate mechanisms for certain environments. The same argument is valid for all other IKMS sub-blocks too.

An information collection approach we consider here is to use a number of distributed information collection points deployed in the different NEMs. The Information Collection Points (ICPs) act as sources of information: they monitor hardware and software for their state, present their capabilities, or collect configuration parameters. The IKMS supports three types of information collection queries coming from the ICD to a NEM ICP(s): (i) 1-time queries, which collect information that can be considered static, e.g., the number of CPUs, (ii) N-time queries, which collect information periodically, and (iii) continuous queries that collect information in an on-going manner.

ICPs should be located near the corresponding sources of information in order to reduce communication overhead. Filtering rules based on accuracy objectives should be applied at the ICPs, especially for the N-time and continuous queries, for the same reason. Furthermore, the ICPs should not be many hops away from the corresponding IKMS nodes (e.g., in case of a distributed IKMS infrastructure). For example, in case of an information aggregation process, the network overhead between the ICPs and the Information Aggregation Component should be minimum.

An ICP can have 5 main components: the sensors, a reader, a filter, a forwarder and an ICP controller. These are described below.

The sensors can retrieve any information required. This can include common operations such as getting the state of a server with its CPU or memory usage, getting the state of a network interface by collecting the number of packets and number of bytes coming in and out, or getting the state of disks on a system presenting the total volume, free space, and used space. In our implementation, each sensor runs in its own thread allowing each one to collect data at different rates and also having the ability to turn them on and off if they are not needed.

The reader collects the raw measurement data from all of the sensors of an ICP. The collection can be done at a regular interval or as an event from the sensor itself. The reader collects data from many sensors and converts the raw data into a common measurement object, consistent to the UniverSelf information model and/or an intermediate format for the channel communication (i.e., to minimize overhead).

The format may contain meta-data about the sensor and the time of day, and it contains the retrieved data from the sensor. The filter takes measurements from the reader and can filter them out before they are sent on to the forwarder. Using this mechanism it is possible to reduce the volume of measurements from the ICP by only sending values that are significantly different from previous measurements. By using filtering the ICP produces less load on the network. In our case, the filtering percentage matches the accuracy objective of the ICD sub-block requesting the information.

The forwarder sends the measurements onto the network. The common measurement object is encoded into a network amenable measurement format. The measurements are encoded using XDR [17] as a way to minimize the size of the transmitted data. The XDR format is commonly used in monitoring systems [18] in order to reduce network loading.

The ICP Controller controls and manages the other ICP components. It controls (i) the lifecycle of the sensors, being able to turn them on and off, and to set the rate at which they collect data; (ii) the filtering process, by changing the filter or adapting an existing filter; (iii) the forwarder, by changing the attributes of the network (such as IP address and port) that the ICP is connected to. These parameters should meet the information collection requirements coming from the ICD IKMS sub-block.

The IKMS is aware of the information collection constraints of the NEMs. The information collection constraints are being communicated during the NEM registration process, which includes an equivalent registration of the NEM as an information source. The ICD supports quality enforcing functionalities using filtering and/or information accuracy objectives, in order to meet such constraints and the global performance goals coming from the Governance.

Furthermore, the IKMS infrastructure can collect information from the NEMs producing information, using different information monitoring techniques. An example is the VLSP infrastructure which was implemented in the context of the UniverSelf.

In the UniverSelf project, we consider the pub-sub information/knowledge querying method as an important functionality that can synchronize communication between NEMs, the Knowledge block and the other UMF core services blocks. With the publish-subscribe type of systems it is possible to achieve fully distributed system for collecting the information [19]. It is also easy to keep an up-to-date list about what information is available from which entities and the associations between the information sources and information users entities. The information about the available information and entities and their associations should be also one input for building the large-scale knowledge management. The simplicity of the xml-based communication protocol of such schemes provides an abstract layer, while a rule description (policy) language can be used to introduce observation capabilities (monitoring). Examples of pub/sub implementations that are already part of the UniverSelf demos are the Distributed Decision Engine and the Siena infrastructure.

Collection, retrieval and querying of context and knowledge of the network can be achieved by employing service discovery protocols. These protocols are tailor-made for acquiring service specifications existing in the system; however their use can be extended for discovering the context information. There are a number of criteria in comparing discovery protocols but they can also apply for discovering context/knowledge. Aspects on the discovery protocols can be summarized to:

- Whether it is Directory-based or Directory-less
- The level of scalability it supports
- Level of stability of the structure

We discuss several approaches to information/knowledge dissemination below. For example, information diffusion could be used:

- ***Gossiping.*** Gossiping, also known as epidemic communication, is aiming at spreading information in order to obtain an agreement about the value of some parameter. Gossiping algorithms are mainly based on the assumption that data are randomly propagated in a network of nodes where such a random propagation can be achieved using specific contact lists per node [18][20]. Thus, each entity interacts only with a few nearby neighbours (e.g., determined by an overlay or by a communication wireless protocol): it can either establish one interaction at a time, or broadcast the information to all its neighbours. Each entity passes its belief of value of the parameter to (some of) its neighbours; when an entity receives such values from its neighbours, it processes and combines them with its current belief. Then it contacts its neighbours to (re-)broadcast to them the newly computed value.
- ***Random choice***. Random choice is used break symmetry among a cluster of elements of the same type, allowing the element to differentiate their behaviour. Moreover, random choices could be performed during the execution of algorithms, so as to select one of the neighbours linked with an overlay. Random choices could be represented by non-deterministic selections during the execution of

the (non-deterministic) automata describing the behaviour of the element or through a random selection of a (numeric) value.

- ***Fields***. The value of a local variable of a computing element in a system (e.g., the ensembles of sensors or of mobile devices) can be considered as a value of a field over the discrete space occupied by the system. If the density of elements is large enough this field of values may be thought of as an approximation of a field on the continuous space. The single elements can process the values in the field. For instance, through gossiping-based protocols, each particle must repeatedly update the value of the solution to be the average of the values sent by its neighbours: this algorithm will eventually converge to the average of all the values in the fields. Moreover, they could apply algorithms for solving differential-equation on a discrete field in order to estimate the evolution of the values. Fields are also related to other bio-inspired primitives, such as gradient and reaction-diffusion.
- ***Gradients***. The gradient primitive takes inspiration from the chemical-gradient diffusion process that is crucial to biological development. Gradients imply the estimation of the distance (and the direction and, then, the path) from each element to the nearest component designated as a source. Each element could store in a local variable the estimated value of the distance, and, so, they can create a field.
- ***Reaction-diffusion***. Reaction-Diffusion primitive describes the evolution of a field, according to a biological and/or chemical metaphor. In principle, a system can be seen as ensembles of elements each of which performing some type of function on the field value (i.e. reaction) and interacting with other elements by means of some communications protocol (i.e. diffusion).
- ***Store and forward***. Asynchronous data-information diffusion can also be achieved by using smart communication primitives that implement store and forward mechanisms both in the producer's and consumer's processes. Communication appears asynchronous and anonymous to the application without the need for an intermediary entity.

Aspects that are important for the information dissemination and the relevant discovery underlying mechanisms are summarized below:

- ***Utilisation of communication means***. Unicast is the most common form of communication in discovery protocols. The sender explicitly addresses the receivers, to which sends the data (query, reply, announcement, etc.) through the network. UDP multicasting is another form of communication. In this approach, a number of nodes form a multicast group by sending a few initiative unicast messages. The last form uses Link Layer broadcast. In this approach, a packet is sent to every node in the vicinity, e.g. within one hop or container network domain.
- ***Discovery scope***. Discovery messages should be limited from unnecessary distribution over the network. By defining proper scopes, unnecessary processing on context clients, providers and information bases are minimised. Scope definition can be based on user rule, other context information and network topologies. Advertisement/ replying policy: discovery protocols are different in replying method to the queries or making announcements to the network. In replying the queries, the context providers may reply to any query they receive, regardless of being necessary or not. In making the announcements, they may also send periodic advertisements to the network, not caring how clients are interested in receiving the adverts. This is called blind advertisement. The benefit of blind discovery is simplicity at the cost of redundancy.
- ***Retransmission policy***. Discovery protocols use retransmission of advertisements or queries in different situations. Retransmission of an advertisement can be retried for emphasising the advert and assurance on receipt of adverts by the clients, or for refreshing and updating context information. Refreshing advertisements are usually less frequent than retrying ones. Retransmission of advertisement can stop after a certain time, and interval between the advertisements can vary.

4.2.2 Information storage mechanisms

There are some issues that should be considered in using the IS:

Centralised/ distributed (flat)/ distributed (hierarchical). The directory-based discovery structure is categorised into centralised and distributed directory. In centralised directory mode, only one directory exists in the network, whilst in distributed directory, several nodes maintain the information. In the distributed mode, the nodes form an overlay network. If the overlay network is formed on a peer-to-peer basis, the structure is called flat, otherwise if it is structured as a tree, is called hierarchical directory.

Context information state. Information states are usually divided into soft and hard states. A soft state context expires after a defined lifetime. After the lifetime, the context information either expires (in which the information bases delete it from their list), or re-announces its presence, then the information bases and clients refresh their cache/directory. In contrast, a hard state context should be periodically polled by the clients and the information bases for checking its availability.

Cache management. Cache containing context information can expire with time, based on the lifetime of the context and change of context. As an example, possibility of unavailability of context info in a network can cause expiration of the cache contents. Cache management system can deal with invalidity of data in different ways; it can either remove the data from cache, or mark it as expired, asking for an update from the context providers. In addition, context providers should refresh context specification prior to expiring lifetime of the context. Choosing proper timeout intervals for refreshing and expiring the cache contents is very important, since improper setting the values can cause resonance and instability in the system. If the cache expires before receiving a late refreshment message, it either deletes the info or sends an update request, whilst proper on time sending an advertisement can prohibit unnecessary cache entry deletion or updating requests.

We are looking at the MongoDB approach for the information storage, which has unique features such as:

- object persistence layers
- automatically providing historical details for the stored information.
- Supports a geo spatial element, etc.

Another important aspect is to synchronize the information base with the pub-sub mechanism. For example, a solution relevant to the Distributed Decision Engine pub/sub approach [28] is capable of caching a short-term asynchronous dynamic information (events) over the network. The main responsible entity for caching functionality in DDE is EventCache(s), which performs, in addition for caching, event distribution and filtering functionalities. Typically in DDE, EventCaches are parallel connected to formalize a distributed information (event) caching system. Incoming event at EventCache is always cached (and forwarded towards the possible interested consumers entities). In addition, every event has a certain time-to-live value (defined by the information origin), after this predefined time, if not replacing event received before, event become invalid. If incoming event's certain properties (origin, event identifier and type) are the same as for already existing event at EventCache, the old one will be replaced with the new one. The producer responsibility is to update the event before the time-to-live value expires.

In case we decide that the IKMS should guide a direct NEM to NEM communication, distributed information storage may have peripheral nodes inside NEMs or a directory service may be looked up from a NEM to locate a specific piece of information in another NEM, before the direct communication takes place. This aspect is currently an open issue.

4.2.3 Information processing and knowledge production mechanisms

A main aspect of information processing is information aggregation. An efficient approach is to use a distributed Information Aggregation (IA) sub-block that consists of a number of nodes: the Information Aggregation Points (IAPs).

The IAPs apply aggregation functions to the collected information (e.g., measurements). The aggregation process increases the level of information abstraction, thereby transforming the data into a structured form, but at the same time reducing the load on the network. Aggregation works in situations where acting NEMs do not need a continuous stream of data from a NEM that produces information, but can get by with an approximation of the data. For example, getting an occasional measurement with the average of the volume of traffic on a network link may be enough for some applications. Some common aggregation functions include SUM, AVG, STDDEV, MIN and MAX.

A structure of an IAP could have 7 main components: a collector, an aggregation specifier, a selector, an aggregator, a filter, a forwarder and an IAP Controller. All of these components are described below.

The collector collects measurement data from the network and converts the measurement format (e.g., XDR encoded) into a measurement object (consistent with the UniverSelf information model). After this the measurement objects are saved in a local NEM data store for later aggregation processing. The local data store can use the Timeindexing Framework [20][21] which allows any kind of data to be stored and retrieved using timestamps or time intervals. The Timeindexing Framework provides the mechanism by which arbitrary sequences of measurements can be selected and aggregated. It creates an index into the data, called a time index, and provides an API for accessing the data.

The aggregation specifier, the selector, and the aggregator are actually combined into an aggregation engine. The aggregation specifier specifies when the aggregator executes, what it aggregates, and how it aggregates. These three specifications are similar to those used in SLA compliance systems [22], because the process of analysing the data is similar.

The when specification is of the form: wake up every N seconds, which will cause the aggregation engine to wake up regularly to provide an aggregation. The “what” specification takes the form of a time interval, such as “from now, back 30 seconds”. The “how” specification is the name of a function to aggregate the data, such as AVERAGE, SUM, etc.

The selector selects the required measurements to aggregate using the "what" specification. It determines what data is eventually chosen by applying the time interval, such as “from now, back 30 seconds”, to the time index and selecting the relevant measurements. In this case, it will cause the selector to select the most recent 30 seconds worth of data. As the data store uses time indexing, the time interval can be changed arbitrarily. Once the selection is complete the selected data is passed to the aggregator.

The aggregator aggregates the selected measurements presented by the selector. It uses the how specification to aggregate data. Although it is most common to use aggregation functions, such as SUM, AVERAGE, STDEV, MIN and MAX, the IAP Controller can pass in an arbitrary function into the aggregator in order to do the aggregation.

This gives considerable power and flexibility when determining aggregations. Once the aggregation is calculated, the aggregated measurement data is passed to the filter. The filter takes measurements from the aggregator and can filter them out before they are sent on to the forwarder. Again, this reduces the volume of measurements by only sending values that are significantly different from previous measurements. Using filtering in this way in the IAP, like filtering in the ICP, less load is produced on the network.

The forwarder sends the aggregated measurements onto the network. The common measurement object is encoded into the same network amenable format as in the ICP (e.g., the XDR). By having the same network format, the consumers of the measurement data do not need to know if data has come directly from and ICP or has come from an IAP. This allows hierarchies of elements to be composed as an IAP can further aggregate data from other IAPs, if this is required.

The IAP Controller controls and manages the other IAP components. It controls (i) the collector, by changing the attributes of the network that the IAP listens to, (ii) the aggregation process, by managing the aggregation engine and by passing in the aggregation specifier, (iii) the filtering process, by changing the filter or adapting an existing filter, (iv) the forwarder, by changing the attributes of the network (e.g. IP address and port) that the IAP sends to.

The aggregation engine itself is flexible enough to be given different aggregation specifications by the IAP Controller in order to process the data in varying way. For example, it can be configured to wake up once an hour and select data for the last day, and then apply an aggregation function. This is achieved using a mechanism that relies on plugins. These plugins represent code blocks, which can be pre-defined, such as an average aggregator, or can be defined to suit the need.

In practice, the IAP controller is handled from the different IKMS sub-blocks or the other UMF blocks. For example, new aggregation functions could be specified from the Governance block, accuracy objectives or filtering could be specified by the IFO sub-block etc.

Information dissemination is done through the ICD. As a NEM may request a specific piece of information from the IKMS, the deployment location of the IAPs should also consider the locations and the traffic requirements of the NEMs retrieving information. As well as requesting information, a NEM has the option to subscribe to a pub/sub based information dissemination service by setting an appropriate threshold to a specific type of information. Whenever this threshold is exceeded, the application is notified.

4.2.4 Information flow optimization mechanisms

The IFO sub-block optimizes information flow using a number of optimization algorithms. These algorithms can potentially implement a variety of optimization tasks that involve performance related tradeoffs. Example algorithms include (i) optimizing with respect to the network protocol deployed, or (ii) trading processing cost for communication cost by using compression techniques. The optimization considered in this document, however, concerns the placement of the IAPs in the network. This is a process, which is carried out when the IKMS infrastructure is initially deployed, but can also be triggered at run-time to react to changes in the network, or to emerging requirements coming from the Governance or Coordination blocks.

The IAP placement is carried out by a novel placement algorithm for dynamic networks, called Pressure which greatly improves performance compared to other proposed placement algorithms. Pressure is a simple and locally optimal greedy algorithm that minimizes traffic overhead. This algorithm is combined with a system for predicting the lifespan of nodes, and a tuneable parameter is also given so that a system operator could express a preference for elected nodes to be chosen to reduce traffic, to be “stable”, or some compromise between these positions. The combined algorithm called PressureTime is lightweight and could be run in a distributed manner.

The Pressure algorithm is a locally optimal, greedy algorithm for placing a new management node in order to reduce network traffic. By locally optimal it is meant that each single node selected is the optimal node to reduce network traffic at that time but this does not account for the future evolution of the network or future nodes which may be selected. A variation of Pressure, also used in this NEM, is called PressureTime and combines the Pressure algorithm with a tuneable life-time maximisation algorithm that attempts to select nodes based on their expected remaining lifetime.

The desirable properties of selected nodes are:

- only a “small” subset of nodes are selected,
- nodes are not “too far” from their nearest leader and hence traffic over the network is minimised, and
- nodes which are selected will stay selected for a reasonable period of time before they either “die” (are deactivated or moved to a different part of the network) or are deselected.

Long-lived nodes are desirable because selecting nodes for management or for data collection will not be effective if the selected node disappears from the network soon afterwards.

The twin objectives of the algorithms described here are (i) to select nodes which reduce management traffic on the network and (ii) to select nodes which exist for a long period of time. Instead of creating an objective function that is a weighted sum of these objectives, the approach taken here is to investigate tuneable trade-offs. Using this method, the network manager could choose a node selection policy which is efficient in terms of management traffic, or in terms of management node stability, or in terms of some combination of these aims, as appropriate. More details on these algorithms can be found in the papers [23][24][25][26][27].

4.3 Coordination mechanisms

4.3.1 Optimization and conflict avoidance mechanisms

As already mentioned, the role of the “Optimization and conflict avoidance” function is to guide the re-computation of the resource allocation to the NEMs in a way that optimizes the global system’s utility, capturing even the end-to-end optimization of different segments and for the detection and avoidance of conflicts between NEMs. The lack of this function may lead not only to sub-par performance but also to unstable and oscillatory behaviours.

A number of mechanisms can be considered in the context of this function with various levels of complexity and intelligence, which is instructed by both the nature of the NEMs that are to be coordinated and also their capabilities in terms of providing the required inputs to the coordination mechanisms.

One key factor that influences the selection and applicability of the appropriate coordination mechanism is the timing of NEMs leading to a category of mechanisms that are based on the “separation in time” strategy. The separation in time strategy in principle dictates that conflicting NEMs should not be allowed to execute simultaneously their enforcements to the network.

For NEMs that have similar time scales this translates into mutual exclusion strategies, where only one NEM at a time is allowed to execute and enforce its actions. In the simplest form this can be implemented by a random token passing mechanism, where the selection of the NEM to “run” is very simplistic without taking into account network performance objectives. This method, however, even though simplistic it offers the advantage that poses very minimal requirements into NEMs in terms of having to be able to predict the outcome of their actions.

A more sophisticated application of a separation in time strategy is through the incorporation of utilities and performance objectives in the token assignment decision. This means that all NEMs that are due to “run” at a certain time point are able to predict the outcome of their actions (if they were to “run”) and the token can be assigned not randomly, but each time to the NEM whose action is expected to maximize the network utility at that time point.

Separation in time strategies can also be applied in the case of NEMs (processes) that have different time scales. In such cases, processes that are optimized rather infrequently set the standard for those processes that are optimized rather frequently. The fact that the slower moving processes are masters with respect to the faster moving processes is a natural choice as slower moving processes are not agile enough to quickly react to changes. It is worth noting that that a separation strategy does not necessarily have to be with respect to time. It is also conceivable that processes where parameter changes are more costly, e.g. because reconfigurations of parameters require a lot of effort, are masters with respect to those processes where reconfigurations of parameters can be achieved rather easily. Yet another strategy makes the choice of separation dependent on the measurements that are needed in order to make a statistical meaningful decision whether a parameter configuration should be modified or not. Last but not least, it can also be helpful to group optimization processes topic-wise as here obviously the likelihood of having a strong coupling is much higher than with unrelated processes.

For NEMs that do not follow a strict cycle (e.g. non-periodic NEMs triggered by certain events during runtime) the token passing mechanism may be requested by the NEMs themselves with the decision on the assignment of the token or not depending –as before- on the intelligence of the underlying coordination mechanism, their features (e.g. expected convergence time) and the capabilities in terms of predicting the outcome of their actions on network performance.

Contrary to separation in time strategies, an alternative approach to the coordination problem is to try and find a compromise in NEMs actions that maximizes an objective function indicative of the network performance. That is, NEMs are not mutually excluded from running but they are considered at the same time. However, their actions are coordinated so that they are not selfish and possibly even contradicting but they complement each other in the best possible way.

A straightforward solution for this would be to integrate their objectives into one optimization function. In this way, the common function will handle the conflicts of the two or more, maybe competing, objectives. A well elaborated approach to do this is through multi-objective (MO) optimization. There are several methods to solve a multi-objective problem. Some classical methods consist of converting the MO problem into a single objective (SO) problem by either aggregating the objective functions or optimizing one objective and treating the other as constraints.

In the case where the objective functions represent performance indicators of a network obtained through measurements, the objective functions will only be known up to a random measurement error, which decreases when the measurement interval becomes larger. In such cases, notions from stochastic optimization (stochastic approximation) can prove beneficial into solving the optimization problem.

It is worth noting that optimization in the case of NEMs/processes operating at different time scales is also possible. A particular case of interest is the hierarchical approach. Say that there are two utilities: the first utility function is tuned on a fast time scale, so that the second can be considered quasi-static, and the second is tuned on a slower time scale, so that the first utility always appear to have converged to its optimal value. The approach easily extends to an arbitrary number of utilities as long as each of them has it’s time scale, and a hierarchy (from the fastest time scale to the slowest) exists. The hierarchical approach is also linked to situations where several agents have different objectives, where one agent is a leader, and the others are followers. The optimal points are known as Stackelberg equilibriums in the context of game theory.

5 Standardization aspects

Sections 2, 3 and 4 provide the necessary definitions, processes, tools and methods for UMF, in order to achieve unification of diverse autonomic solutions, governance of automatically managed infrastructures and services, and "plug and play" of autonomic solutions within existing and future management ecosystems. Despite the soundness of UMF vision from the research point of view, careful and well planned roadmap towards standardization is required in order to boost its deployability and operator adoption. The roadmap covers the identification of the parts of the UMF specification that should be standardized, as well as the opportunities for contributions and actions in various relevant standardization bodies/groups, along with possible standardization actions. These standardization efforts will facilitate acceptance and re-usage the research outcomes, and adoption from the telecommunication/networking market.

5.1 UMF and Standardization

The key role of the UMF core entails the need for specification of standardizable interfaces associated with the core components, in order to enable the transfer of the technology/system to marketable product. This specification/standardization process requires identification of the characteristic information and messages to be conveyed from/to each of the individual pairs of entities across the interface, but also and more importantly, it encompasses the specification of the services that each of the block should offer to the rest system (UMF entities or even end-users etc.) i.e. the operations that manipulate these information/messages.

NEM's definition implies that NEM's can be developed by any actor of the telecommunication/networking market: equipment vendor, network management system vendor, network operator, software developers (NEM developers). In order to be ensured operation, interworking and cooperation of the new NEMs in UMF-compliant networks, it is needed the standardization of the respective UMF/NEM specifications. These include prescriptive generic models (i.e. NEM skin) and corresponding interfaces, which will guarantee high reusability, openness and extensibility.

The definition of an information model that will serve UMF's needs/objectives in regard of the modelling of all the business and management aspects and processes (e.g. policies, resources, context information, knowledge) is of high significance for UMF specification completion. Moreover, as UniverSelf targets at unification/federation and end-to-end service management view spanning wireless/wired segments, it is demanded the transformation of vendor/technology-specific management data and languages into a common model used by UMF to perform its management functions. In this context, UMF needs a dynamic, extensible and semantically rich model, in order to facilitate the automated mapping/translation to technology- specific technology data models, merge/consolidate the information gathered from different network domains, and perform advanced autonomic operations including reasoning, learning, and inferring higher level knowledge crucial for the UMF operation.

Furthermore, as one of the main goals of UniverSelf is to demonstrate the reliability of autonomic solutions and develop testing and certification processes, trust issues have to be part of the UniverSelf standardization strategy. In this context, the relevant standardization efforts should cover the specification of the parameters/metrics for UMF efficiency testing/assessing/certifying, as part of the interfaces exposed towards the operator side (vertical trust).

Finally, as UniverSelf use cases represent crucial network problems, they can be submitted to standardization bodies/groups in deriving further requirements and eventually in extending and amending the existing functional design.

5.2 Standardization Opportunities

European Telecommunications Standards Institute (ETSI).

The objective of ETSI AFI (Autonomic network engineering for the self-managing Future Internet) Industry Specification Group [29] is to develop pre-standard relevant specifications, and its operation is driven by three Work Items (WI).

AFI WI#1 is responsible for describing the scenarios and use cases, and defining the key operator requirements that reflect real-world problems and can benefit from the application of autonomic/self-management principles. So, UniverSelf can update these use cases and requirements based on its relevant technical outcomes.

AFI WI#2 aims at designing a Generic Autonomic/self-managing Network Architecture (named GANA) as reference model for engineering the Future Internet. This framework encompasses the specification and design of functional blocks, as well as the characteristic information being conveyed in the reference points among them. Therefore, UMF core functional blocks and associated interfaces can amend this work.

AFI WI#3 is composed of four branches that address different networking technologies and contexts: the NGN reference architecture, the Broadband Forum (BBF) reference architecture, mobile network architectures (3GPP and non 3GPP), and Wireless Ad-Hoc/Mesh/Sensor Network Reference Architectures. In this context, UniverSelf can utilize work on UMF deployment aspects to enrich the relevant activities.

Furthermore, AFI is taking part to the new ETSI restructuring initiative: E2NA (Enhancing ETSI Network Activities). UniverSelf, based on the gained experience of the first two years of research work, can take part in the relevant discussions and contribute to the different aspects.

Finally, UniverSelf's work can be used as base for the creation of new AFI work items in order to tackle other technical topics addressed by the project, such as trust and confidence building in autonomic networks.

TeleManagement (TM) Forum.

TM Forum [30] focuses on enabling service provider agility and innovation, through service creation, management and delivery for providers and operators. From the several evolving standards within its Frameworx program, the most interested initiatives for UniverSelf goals are the Shared Information/Data Model (SID), the Multi-Technology Operations Systems Interface (MTOSI) and the enhanced Telecom Operations Map (eTOM).

The SID aims at providing an information reference model and common vocabulary, addressing both the business and systems perspectives, in order to enable an end-to-end service management. The UMF information model was defined using the SID patterns, allowing information sharing across different layers, administrative domains and network segments. Regarding NEM, as managed element by the UMF blocks, was modelled as new class that is inheriting from the root class within the SID.

The MTOSI is a standardized (XML-based) interface between Operations Systems (OS-to-OS), covering also, as a special case, the Network Management System-to-Element Management System communications (NMS-to-EMS). In this context, MTOSI can be utilized for the interfaces among core UMF blocks and with other legacy systems.

The eTOM provides a multi-layered view, or hierarchical catalogue, of the main business processes in the telecommunication industry. The eTOM was utilized for several aspects in UMF work, e.g. the abstracting three levels for policy refinement. Generally, bringing autonomics through the UMF core will shape the way eTOM operations are realized and maintained.

Consequently, UniverSelf can take part in the discussions of all the relevant groups/initiatives, contributing with the research outcomes (for example with addition of new elements in policy models of TMF Policy Information Exchange group). Moreover, TMF has recently started an ontology program with the aim to explore whether ontological techniques could help to reduce the complexity of and managing and maintaining the Frameworks, their evolution and application. This program will provide OWL definitions for eTOM, SID, linkage between TMF frameworks using ontology and so federation of information, machine readable format of data for reasoning and inference, and as such it is in alignment to the model/functionalities extensions that are developed in the context of UniverSelf.

3rd Generation Partnership Project (3GPP)

3GPP works on a variety of subjects related to radio, core network and service architecture, comprising radio interface, architectures and protocols, strategies of radio resource management, SONs, services, features, management framework and requirements for 3G. Particularly, SON, as part of the 3GPP LTE [31], provides an autonomic management framework for reducing traditional high operational and capital expenditures during the entire network lifecycle. Coordination aspect has become part of the 3GPP standardization agenda in capturing the need to coordinate different SON functions and in particular to prevent or resolve conflict functions i.e. when two or more SON functions try to change the same network configuration parameter or to prevent or resolve negative influences between SON functions. UniverSelf can contribute to 3GPP SON related work with its own development of SON coordination function (developed in the context of one of the project use cases).

Next Generation Mobile Networks (NGMN)

The Next Generation Mobile Networks (NGMN) alliance aims at supporting relevant standardization groups, such as 3GPP and TM Forum, by providing recommendations, requirements and use cases (among other actions). In this context, the Next Generation Converged Operations Requirements (NGCOR)[32] (see section 7), which is a continuation of the projects SON and NGMN Top OPerational Efficiency (OPE) Recommendations and worked in collaboration with TM Forum and 3GPP, aims at describing requirements for converged operations for wireline and wireless networks, providing an significant opportunity for UniverSelf respective contribution, based on the corresponding research outcomes.

International Telecommunication Union - Telecommunication (ITU-T)

ITU-T produces standards for telecommunications, services and internet in the form of Recommendations developed by Study Groups (SG). The most relevant group to UniverSelf work is SG13, which established pre-standardization “Focus Group on Future Networks (FG-FN)” to share discussions on and ensure global common understanding about FNs. FG successfully completed its work in December 2010. UniverSelf/UMF work has already influenced this group and has the opportunity to contribute to the ongoing detailed specification and standardization in the areas of in-network management and virtualization in FNs, as captured by Recommendations ITU-T Y.3001 "Future Networks: Objectives and Design Goals" and ITU-T Y.3011 "New Framework of network virtualization for Future Networks" [33] , respectively.

Internet Research Task Force - Next Generation Mobile Networks (IRTF-NMRG)

The Internet Research Task Force (IRTF) is a major contributor to emerging IETF standards that led to the evolution of the management architectures, models and protocols of the future Internet. Among the existing research groups, the Network Management Research Group (NMRG) is of particular interest for UniverSelf.

The NMRG focuses on higher-layer management services that interface with the current Internet management framework, aiming at identification and documentation of respective requirements, specification of suitable solutions, and proof-ness via prototype implementations, tested in large-scale real-world environments. In this context, UniverSelf can promote its achievements and facilitate their acceptance within the IETF community e.g. for addressing the coupling of self-management features with IETF based protocols and practices. The Internet Draft "A framework for Autonomic Networking” [34] exhibits great commonalities to UMF e.g. in its reference to discovery (knowledge), intent (policies), abstraction levels/autonomic reporting (governance), decentralisation and Distribution/Modularity (NEMs), Life Cycle Support (NEM lifecycle), hence it can provide an opportunity, but also inspiration for contributing elements of UMF into IETF.

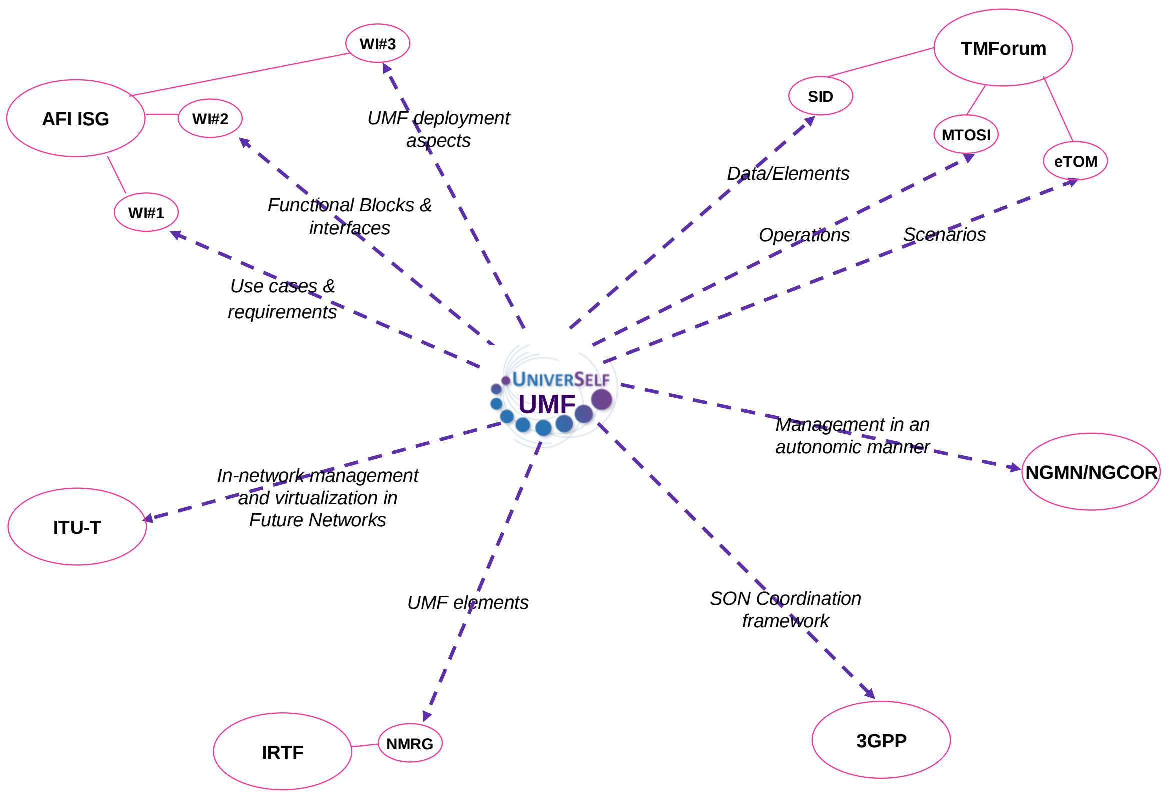


Figure 35. Standardization opportunities for UMF/ UniverSelf.

6 UMF in practice

Use Case 6 is an illustrative example of how UMF is used in practice. According to the scenario, a network operator owns a multi-technology and multi-vendor infrastructure, on top of which he provides a set of applications. The management of both the wireless and core segments of his networks is based on UMF compliant entities, while the operator's interaction with the system is done by using the provided Human to Network (H2N) Graphical User Interface (GUI). This H2N GUI is a powerful and easy to use tool that on one hand enables the operator to model the characteristics of the application, the users or the infrastructure that comprise the service provision environment. And on the other hand, it allows the operator to govern the underlying resources by specifying associations between applications, user classes and quality levels, defining policy rules and registering service provision requests.

More specifically, in UC6 it is assumed that there is a Video Conference application that is available in three quality levels, identified in a high level manner by their names, namely Gold, Silver and Basic. In the same notion, there are also three classes of users with the same names mentioned before. The service and all the relevant details are described in the so called Service Manifest. One of the important information included there is the allowed combinations of user classes and quality levels. In UC6, the users of the Gold class shall receive the service at the highest quality level (Gold). On the other hand, the users of the Silver class can get the service at either the silver or the basic quality level. All these are communicated to the system through the H2N tool in the form of policies and they are stored in the appropriate repositories.

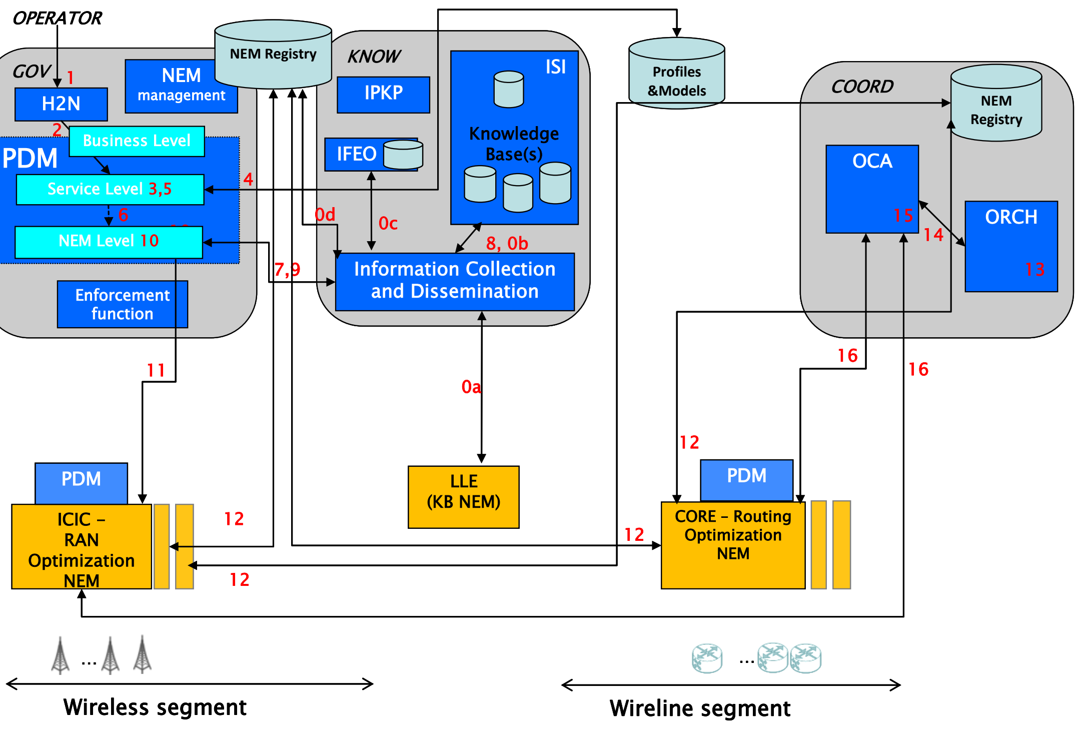


Figure 36. Example based on UC6.

At some point the operator is informed by the sales department, that there will be a press conference in a hotel in the city centre and thus an additional traffic load is expected at this area in the corresponding time zone. The estimation is that 20 Gold users and 15 Silver users of the Video Conference application will be active concurrently, in excess of the usual load. This information (High Level Parameters) is inserted by the operator into the H2N tool (step 1), along with the high level goal of energy efficiency (High Level Objective), based on that a Business Policy is built by the Policy Derivation and Management (PDM) UMF Core mechanism, which in this case has the form of service request (step 2).

Table 10. Abbreviations

Acronym	Meaning
H2N	Human to Network interface
IFEO	Information Flow Establishment & Optimization
IPKP	Information Processing and Knowledge Production
ISI	Information Storage and Indexing
LLE	Load Level Estimation
OCA	Optimization and Conflict Avoidance
ORCH	Orchestration
PDM	Policy Derivation and Management

PDM consists of three levels, the Business, the Service and the NEM level (see section 3.2.2). When the request arrives at the service level of PDM, the first task it performs is the translation of the business level entry request that is received into service level terms (step 3). More specifically, it interacts with the policy repository, the profiles and models repository and it retrieves the quality parameters and the related values or thresholds that should be respected according to the user class (step 4). The Check Feasibility & Optimize operation of PDM analyses the current status of the network and the available resources, as well as the situation (step 5) that will arise after the appearance of this extra load and if there will be significant change in the network conditions. If it doesn't diagnose any potential problems, it forwards the request to the next level of PDM (step 6), so that the appropriate configurations will be derived in order to serve all the users at the appropriate quality levels.

The NEM level of PDM first retrieves information about the infrastructure existing in the concerned geographical area, and then it requests from the Information Collection and Dissemination (ICD) entity, any previous knowledge on the load level of the available base stations (step 7). The underlying core segment routers are not taken into account at this stage. Specifically in UC6, the request is about a hotel located in the City Center area, where there are 3 LTE base stations, BS 16, BS 17 and BS 18.

ICD seamlessly retrieves this information from Information Storage and Indexing (ISI) (step 8). This information may be already stored in the KNOW block. In this case, the knowledge has been sent towards the ICD in order to be stored, i.e., in our example the load level estimation (LLE) NEM has already produced the answer with respect to the load level that will be reached in a specific RAN element at the specific time zone of the request and has already sent it to the ICD sub-block of KNOW block (step 0a) which has seamlessly stored it in ISI (step 0b). If there are two NEMs building the same knowledge, there are two records in the same or different databases and the selection will be done by IFEO through a context aware policy and accuracy objectives (step 0c). In case the information is not available in ISI, then ICD is advised by the NEM registry which NEMs can produce it and triggers the most appropriate (according to the request) for collecting the required information (step 0d). Eventually, ICD sends the retrieved or collected information to the NEM level PDM (step 9).

The Check Feasibility & Optimize operation of PDM in NEM level is then triggered (step 10) in order to decide what kind of optimization should be done for the network to accommodate the requests, namely in this case, the most appropriate solution for handling the new network conditions that will appear due to the additional traffic load. The solution involves a subset of the underlying infrastructure in the area and it mainly concerns the NEMs that manage this infrastructure and are most suitable to elaborate on the solution's details. In other words, the output of the NEM level PDM is a number of policies to specific NEMs, defining at least the portion of traffic that each NEM should undertake and potentially further instructions on the methodology or the objective.

In the examined use case, BS 18 is expected to be anyway highly loaded at that time, so NEM level PDM doesn't assign any more traffic load to it. It chooses to split the additional users as equally as possible to BS 16 and BS 17. So it prepares the corresponding policy and informs the NEMs that are managing these two base stations, namely RAN_NEM_16 and RAN_NEM_17 respectively (step 11). The CORE_NEMs that are supporting them will be notified as well, but there is no need for an explicit action from NEM level PDM about that. In parallel, NEM level PDM passes to the NEMs, through the Enforcement Function, the general objective set by the operator at the beginning about the energy efficiency.

Table 11 UMF CORE Blocks and Functions

Block	Function	Objective
GOV	H2N	Expresses the operator’s business goals (High Level Objectives) and requests
	NEM management	Enables the control of the deployed NEMs and the management of their lifecycle (including the activation and deactivation of the autonomic functionality).
	PDM Service Level	Analyses events and translates service requirements to network conditions
	PDM NEM Level	Identifies potential solutions for operator’s demands/request
KNOW	ICD	Function responsible for activities related to information collection, update, retrieval, dissemination and querying
	ISI	A logical construct representing a distributed repository for registering NEMs, indexing (and optionally storing) information/knowledge
	IPKP	Consists of two components, Information Aggregation (IA) and Knowledge Production (KP). The IA component is a distributed structure that applies aggregation functions to the collected data/information. KP component handles and produces globally-scoped knowledge
	IFEO	Regulates information flow based on the current state and the locations of the NEMs producing information
COORD	ORCH	This function is responsible to address orchestration issues of NEMs. This functionality addresses issues such as ordering the sequence of NEMs in a way that is needed to resolve dependencies in inter-NEM relations based on service/scenario policies from the operator and corresponding input/output and timing relationships, as well as to maintain the proper workflow
	OCA	This function is responsible for guiding the re-computation of the resource allocation to the NEMs in a way that optimizes the global system’s utility, capturing even the end-to-end optimization of different segments and for the detection and avoidance of conflicts between NEMs

The message is sent to the NEMs through a Send NEM Mandate and then, each NEM registers to the NEM registries of GOV, KNOW and COORD (step 12). The ORCH inside COORD identifies that a change in the COORD NEM registry is observed by a change in the instance descriptions of the already registered NEMs of BS 16 and BS 17 (step 13). Thereupon, ORCH triggers the OCA to solve the joint optimization problem between the NEMs of RAN and Core segments, in order to resolve possible incompatibilities between the offered QoS from RAN’s NEMs and core segment’s NEM and to achieve coherence (step 14). Then, OCA chooses the coordination mechanism, checks the feasibility of the mechanism, set the parameters for the selected mechanism and produce the NEMs control policy (step 15) for the NEMs that will need to be controlled by the selected coordination mechanism i.e. the NEMs of RAN_NEM_16 and RAN_NEM_17, as well as the core segment's NEM. The dependency between these NEMs (of RAN and core) is instructed by the ORCH constraints. After a check on whether the mechanism can operate as intended and whether the NEMs can enforce the instructions of the NEM control policy, the NEM control policies are sent to the RAN NEM_16 and RAN_NEM_17, as well as to the core segment's NEM (step 16).

In addition to the control policies, CORE NEM is informed about the specific traffic load that the base stations requested. For instance, in our use case, the control policies indicate that there should be an energy efficient network operation, while BS 16 and BS 17 requested the accommodation of aggregated traffic of about 15Mbps and 20Mbps respectively. Therefore, taking into account this information, CORE NEM evaluates network's status (e.g. utilization of links, consumed energy of activated elements) in order to find the optimal routing configuration. From the evaluation process, it recognizes that traffic is already routed from Video Server 1 towards BS 17 through the links that connect Video Server 1 with LSR 1 and then to LSR 5, LSR 8, LSR 12 and finally to BS 17. Also, traffic is routed from Video Server 5 towards BS18 through the links that connect Video Server 5 with LSR3 and then to LSR6, LSR10, LSR 14 and finally to BS 18. With these currently activated network elements, the most energy efficient solution is to preserve Video Server 1 as the traffic generator and reuse the already established path between Video Server 1 and BS 17 for the new traffic request of BS 17. Regarding the request of BS 16, the optimal solution will be to route traffic generated from Video Server 1 towards the path that traverses routers LSR1, LSR5, LSR8, LSR11 and LSR13. This path activates minimum number of unutilized links, resulting in minimum increase in the consumed energy of the network. Finally, this decision is enforced to the network by sending the appropriate commands to the ingress routers.

Table 12 UMF NEMs

ICIC (RAN Optimization NEM)
The problem addressed by the ICIC NEM is to find the appropriate OFDM resource (subcarriers (SCs) or physical resource blocks (PRBs)) allocation in the target cell (i.e. the cell that this NEM is applied), in order to minimize the interference caused at the target cell's users, by taking into account the target cell context (load, radio conditions etc.), the amount of available resources and the context of the neighbouring cells (acquired by Relative Narrowband Transmit Power (RNTP) signalling) in downlink LTE networks.
LLE (Knowledge Building NEM)
The purpose of this NEM is to provide estimations to other mechanisms regarding traffic load for multiple RAN elements, in specific time periods. Such a goal is achieved through online, unsupervised machine learning schemes. Monitored data are fed into the NEM (by some monitoring NEM) in order to build a knowledge base (which consists of multiple Self-Organizing Maps) containing past experience on traffic load through time for the specific network element.
CORE - Routing Optimization NEM
This NEM provides a solution to the problem of routing optimization with respect to different operator's policies. Our solution is based on a heuristic algorithm that evaluates network's status and finds the optimal routing configuration, exploiting the capability of splitting traffic and forwarding it through different multiple MPLS paths, when this is needed. Main objectives that have been examined are load balancing and energy efficiency. Load balancing is achieved through splitting traffic, while energy efficiency is achieved through the aggregation of traffic into minimum number of links and deactivation of unused network elements. Furthermore, our solution comprises two important features, monitoring network and informing/alerting other NEMs.

Table 13 City Center Infrastructure

Wireless Segment	Corresponding RAN NEM	Wired Segment	Corresponding CORE NEM
LTE BS 16	RAN_NEM_16	LSR13	CORE_NEM
LTE BS 17	RAN_NEM_17	LSR12	CORE_NEM
LTE BS 18	RAN_NEM_18	LSR14	CORE_NEM
LTE BS 19	RAN_NEM_19	LSR13	CORE_NEM
LTE BS 38	RAN_NEM_38	LSR13	CORE_NEM

In the city area, there are 3 LTE base stations, BS 16, BS 17, BS 18 and 14 routers (Label Switch Routers), LSR1-LSR14. Core network has a typical IP/MPLS architecture. Traffic is generated and forwarded, utilizing MPLS paths.

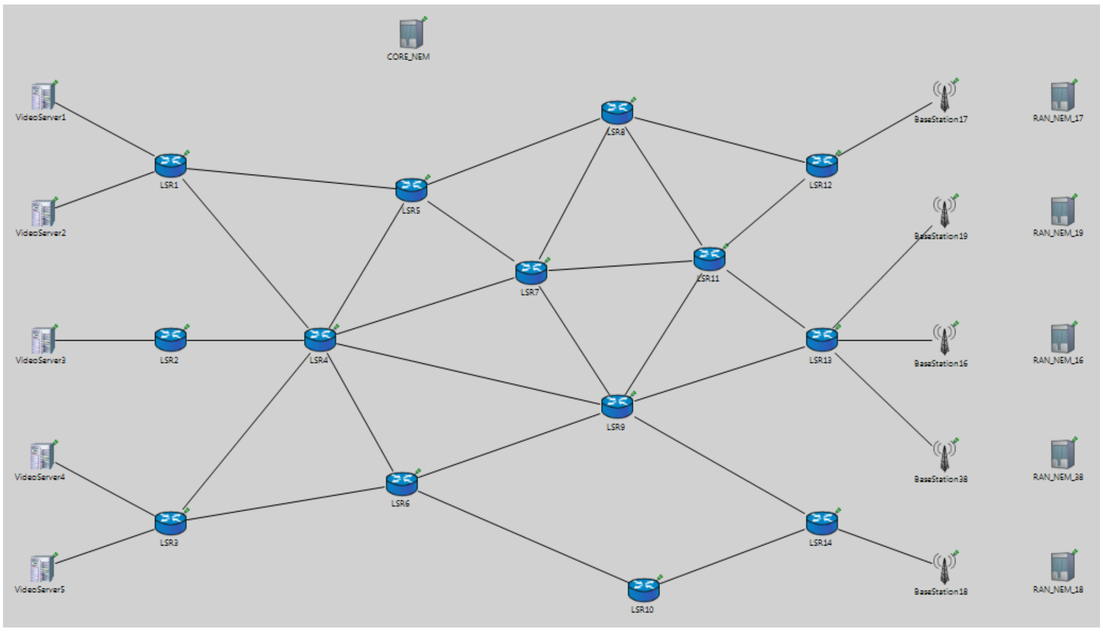


Figure 37. Core Network Topology.

7 Requirements Analysis

The overall UMF requirements list and design goals are derived from the Description of Work (DoW), individual project partners' expertise, as well as the general vision and research directions for Future Networks, Service Oriented Computing and Networking, and Future Internet.

The UMF requirements list has three axes (see Figure 38): a "*bottom-up requirements*" synonymous of 6 use case problem specific requirements addressing operators' day-to-day problems identified in live networks and on existing service/network architectures; a "*top-down requirements*" synonymous of high-level functions, functional blocks and interfaces and "vertical requirements" synonymous of a reposition of TMN FCAPS towards the management functions of Future Networks.

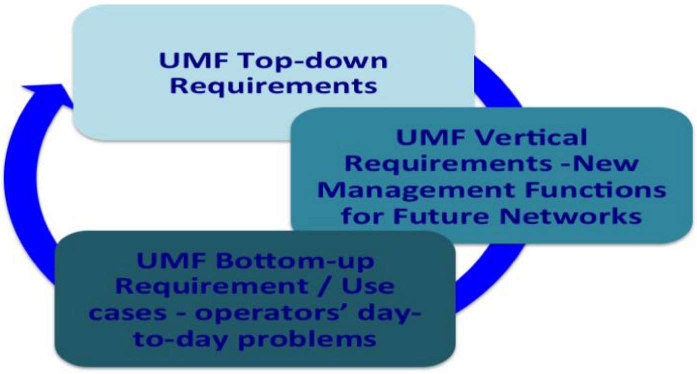

UMF Top-down Requirements
UMF Vertical Requirements -New Management Functions for Future Networks
UMF Bottom-up Requirement / Use cases - operators' day-to-day problems

Figure 38. UMF Axes of Requirements

The first approach "bottom-up requirements" aims at addressing the set of requirements elicited for 6 use cases defined and developed so far within deliverable D4.1 – Synthesis of Use Case Requirements, Release 1 (WP4) and deliverable D4.2 - Synthesis of Use Case Requirements, Release 2 (WP4). The second approach "top-down requirements" aims at addressing global management characteristics across many networking and service domains and they were developed so far within deliverable D2.1 – Unified Management Framework, Release 1 (WP2). The third approach "vertical requirements" aims at elaborating the expected new management functionality of future networks developed so far within deliverable D2.1 – Unified Management Framework, Release 1 (WP2). The requirements together as a set, and not necessarily per individual requirement, describe what distinguishes UniverSelf from earlier network and service management technologies and what the UniverSelf project intends to design and deliver.

The following is a synthesis of the main UMF requirements and characteristics (see Figure 39).

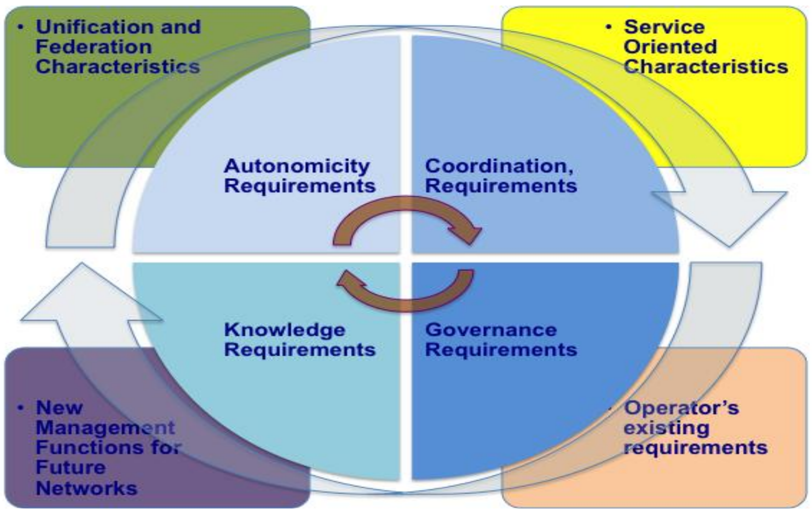

Unification and Federation Characteristics
Service Oriented Characteristics
Autonomicity Requirements
Coordination, Requirements
Knowledge Requirements
Governance Requirements
New Management Functions for Future Networks
Operator's existing requirements

Figure 39. UMF Requirements Synthesis

Unification and Federation

The UMF design aims at an integration and unification of these three axes supporting management operations and functionality by the means of a highly distributed functional architecture. UMF must ensure that multiple diverse management systems implemented upon different autonomic architectures will be able to interoperate and federate. It will also guarantee that autonomic functions may be implemented independently of the architecture chosen for the management system. As such UMF is envisaged as a multi-faceted unification: a unified and evolvable framework constituting a cross-technology (wireless and wireline) and common abstraction/substrate for supporting the management of both networks and services.

Management processes and functions can be implemented as external and separated, or inherent management capabilities of the network or services. The main objective is the design of UMF management functions that are located in or close to the network elements and services to be managed, in most of the cases co-located on the same nodes e.g. embedding management capabilities in the network. The main benefit of the resulting architecture is the inherent support for self-management features, integral automation and different degree of autonomic capabilities, easier use of management tools and empowering the network with inbuilt cognition and intelligence. Additional benefits include reduction and optimisation in the amount of external management interactions, which is key to the minimization of manual interaction and the sustaining of manageability of large networked systems and moving from a managed object paradigm to one of management by objective. Key supplementary benefits include also the unification of intelligence that allows the system to govern its own behaviour in terms of network and service management and unification of network orchestration that enable cooperation and interworking of closed control loops specific to different management functions and operations.

UMF Decomposition and Extensibility

The analysis of all requirements, “bottom-up”, “top-down” and “vertical” requirements, have resulted in the definition of a set of UMF functional blocks and interfaces that consider both services and networks and exhibit the flexibility to accommodate mixed networking scenarios spanning both wireline and wireless technologies. In addition the resulting UMF functional blocks are grouped in *Core functions*, which are supporting all UMF functions, and *Network Empowerment/Intelligence functions* which are acting and changing groups of network, computation and storage physical and virtual resources. Each UMF Network Intelligence functions would encapsulate at least one self-x algorithms/methods and it will host /deployed by the network in case of in-bound management) or by TMN/TMF Network Management Station (NMS) / Operations Systems (OS) in case of out-of-bound management). Such Network Intelligence functions retrieve data from network/service elements and agents for the purpose of monitoring and controlling networked devices and make changes to the following managed physical and virtual entities:

- Services: Large number of ICT and Telecom services offered by the network operator or different service providers needs to be managed (e.g., management of the mapping of service components into executable services on the network environments, deployment and activation of services, services run, the service profile/requirements, manage the e2e performance of the services, assurance management, charging/accounting management, etc.)
- Networks: Different technological (e.g., wired, wireless), topological (e.g., enterprise, access, core) and administrative domains need to be managed (i.e., enforce policies, configure components, monitor management data, etc.)
- Resources: The per node computational resources (e.g., buffers, memory, CPU), network resources (e.g., spectrum, radio channels, network interfaces, etc.) as well as virtual resources, which are dynamically created groups of physical resources need to be managed in an autonomous or cooperative way.
- Domains: A grouping of resources and managed objects with uniform set of policies (e.g. administrative domain, access-network domain, core network domain, virtual network domain, service domain, etc.).
- Managed Things: S/W objects, which are part of management applications/services, Virtual Machines representing service components and virtual routers, network attachments, domains, smart objects / Internet of things.

The *Core functions* are mainly derived from the top-down requirements and they are further grouped in *Governance, Knowledge and Coordination functional blocks* capitalising on previous autonomic architecture research as a coherent set of autonomic management functionalities that can interwork in a scalable manner.

The eco-system of *Network Intelligence functions – the Network Empowerment Mechanisms (NEMs)* - include the functions resolving operators' day-to-day problems identified in live existing service/network of the identified 6 operator's existing day-to-day use cases and the supplementary functions of managing future networks.

UMF can be extended mainly via additional NEMs or through modification of existing NEM functionality and characteristics, while minimizing impact to existing system functions. The degree of extensibility covers Plug_and_Play/Unplug_and_Play approaches, on demand deployment of new management functionality and dynamic programmability of management functions.

Service Orientation

Much related to unification above is the service orientation of UMF. UMF will be service oriented and will offer a service view instead of the traditional resource view. This means that UMF should cover explicitly both network and services aspects in a unified manner and facilitate shifting and convergence towards "Everything as a managed Service", which also includes "Network as a Service" (e.g. management of the integration of network and service aspects).

Autonomicity and Self-x

Autonomicity/automation and self-x networking are of topmost importance for UniverSelf and they should be facilitated by and demonstrated through UMF. A number of coordinated, autonomic, closed control loops per management function or group of management functions will need to be specified. In particular, UMF should provide a framework for understanding the behaviour of active self-x entities. It should be also able to assess their performance and when needed i.e. at ideal points in time, to re-optimize individual management processes. This last might also designate the need to satisfy extensibility (change of management functionality) requirements. That is, UMF must provide the enablers for activating new management functionality on demand in a plug-and-play / unplug-and-play fashion and programmatically, but also the capability to adapt the information flow and interactions between the functions of the UMF to face new system or operational requirements.

Governance

The prominent role of governance in UniverSelf calls for explicit design of its management functionality and associated interfaces within UMF. First of all, the UMF design should designate and facilitate the development of a privileged, powerful and evolved human to network interface that will be used by the human operator for expressing their business goals and requests, thus shifting from network management to network governance. At the same time, UMF should provide a policy-based framework for translating those business level goals/requests (highest level policies) to low level policies and configuration commands. In general, UMF must facilitate high-level dialogues between self-managed networks and multiple human network operators. They will ensure that all well-formed queries to the network are answered in a pertinent way and also that every well-formed goal injected to a network is either enforced completely and instantly or its delay/modifications are negotiated per rules instantiated. In the opposite direction, UMF must take care so that every context to continue self-managed operation or realistic danger of that will be reported to humans with pertinent details of the situation. Having a global coarse view of the network components and services, governance participates in the overall evaluation on the performance of services/network nodes/domains etc.

Coordination

In supporting autonomicity above, UMF should also provide a framework for the coordination and orchestration of the newly introduced self-x managing and managed entities. This can be based both on human control/directives (i.e. governance) and explicit functionality destined to this task. Additionally, this introduction of autonomic/self-x network capabilities into a network and services might cause instabilities, thus jeopardizing performances and integrity. Therefore, UMF must provide the means to monitor, detect/predict, resolve and manage external/internal disturbances/dynamics in networks and services.

Knowledge

In supporting autonomicity above a unified Information and knowledge management system is envisaged. It is a critical part of the UMF since it plays the role of information & knowledge collection, aggregation, storage/registry, knowledge production, distribution and optimisation across all UMF functions and functional blocks.

New Management Functions specific to Future Networks

UMF will capitalize both on research done in autonomic networking and demonstrate its applicability to industry standards, whereas at the same time it will be forward looking, enabling future research and engineering to build on UniverSelf outcomes. The top level requirements regarding management of future networks that follow, were actually identified by ITU-T SG13 “Focus Group on Future Networks (FG-FN)” and are expected to play quite a role in the finalized UMF design and in demonstrating its future-proofing. New management functions envisaged for Future networks are (see Figure 40): I. Service awareness management functions including management of service diversity, functional flexibility and programmability, management of virtualisation of resources, in-network management enablers, management of mobility and management of reliability; II. Data awareness management functions including data and context access and data identification; III. Environmental awareness management functions including energy management and multi-objectives optimisation; IV Social and economic management functions including management of service universalization and economic incentives.

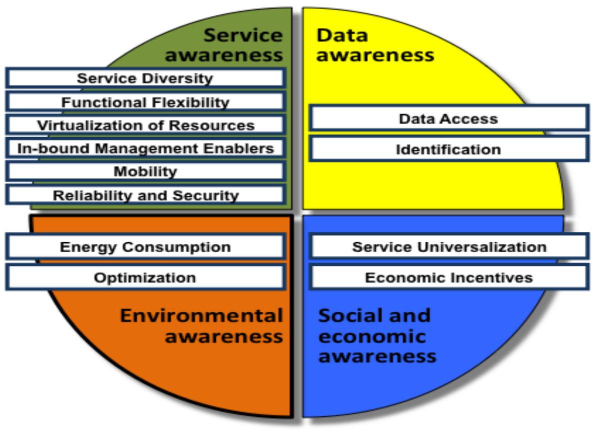


Figure 40. New Management Functionality for Future Networks

8 Conclusion

Deliverable D2.2 "UMF Specifications – Release 2" provides a first complete functional specification of the UMF, including the detailed specification of the UMF core and the relevant interfaces, the possible mechanisms to support the main functions of the core blocks, and the demonstration of the realization potentiality of UMF.

In this context, a set of definitions for the NEMs were introduced and used for the full description of the NEM's lifecycle. For the core blocks, the mandatory and optional functional blocks were determined. Specifically, the expected role, the behaviour and the functions of each core block were described, and the structure and the relevant interfaces between the functions of each component were determined. The related information was prescribed, comprising the operation description, the necessary constraints, list of input and output data, as well as the list of non-functional requirements. The interfaces between the three core components were described and possible mechanisms to support the main functions of the core blocks and achieve their objectives were identified. Furthermore, activity diagrams that provide a dynamic view of UMF operations were defined.

The UMF information model was defined by refining and extending the TMF information framework (i.e. SID) patterns. The classes, attributes and relationships of the UMF information model allow information sharing across different layers, administrative domains and network segments. Moreover, the UMF capture for the UC6: Operator-governed, end-to-end, autonomic, joint network and service management, featured the potentiality of realization of UMF.

The UMF specification will be refined, updated and consolidated as the research highlights and clarifies the issues, and feedback from the integration of the network empowerment solutions (WP3, including already first examples in deliverables D3.5, D3.6/D3.7 and the upcoming D3.8 where the NEMs are examined not only in a stand-alone way, but also as part of typical interactions involving the UMF capabilities) and feasibility/implementation of the UMF (WP4, including already first examples with the release of the two first project prototypes (deliverables D44 and D48 along with their corresponding leaflets (D45/D49 respectively). The UMF aspects are also taken into consideration in the deliverable D46 which reports on the deployment assessment of the project solutions) will continue and be published as Deliverable D2.4 "UMF Design – Release 3". This final version of the UMF will accommodate requirements from all use cases handled by the project and will incorporate corresponding network empowerment solutions for Future Networks as applicable to the overall networking infrastructure, spanning wireless and wireline, as well as access, core and service segments.

Capitalizing on this second UMF specification, next steps include: a) detailed specification of the UMF core mechanisms, b) accommodation of further, future use cases as a means to prove a great level of reusability of functional blocks and/or interfaces, c) validation activities, d) standardization activities, and e) instantiation of UMF mechanisms within exemplary use cases, e) system architecture assurances that would make UMF ready for deployment with a migration path.

As the final specification of the UMF has to enable standardization and certification, in order to ensure industry adoption, the UMF standardization activities will concentrate significant effort. The UMF standardization strategy comprises activities related, for example, to 3GPP subjects (e.g. architecture, features and requirements for SON mechanisms and SON coordination, specifications of measurements related to SON, OAM aspects and use cases related to SON, system architecture and service requirements for future mobile networks, as well as system enhancements for autonomic load balancing of core network gateway and nodes and for autonomic energy saving solutions), to Next Generation Management Networks – NGMN (e.g. use cases and definition of OAM requirements), to ETSI AFI Industry Specification Group (e.g. scenarios, use cases and requirements for Autonomic/Self-Managing Future Internet) and ITU-T Future Networks Group.

9 References

[1] Strassner, J.; Yan Liu; Jiang, M.; Jing Zhang; van der Meer, S.; O Foghlu, M.; Fahy, C.; Donnelly, W.; , "Modelling Context for Autonomic Networking," Network Operations and Management Symposium Workshops, 2008. NOMS Workshops 2008. IEEE , vol., no., pp.299-308, 7-11 April 2008, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4509963&isnumber=4509916

[2] Steven Davy, Brendan Jennings, John Strassner. The policy continuum—Policy authoring and conflict analysis. Computer Communications 31 (2008) 2981-2995.

[3] 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 InternationalWorkshop on Distributed Systems, Operations and Management (DSOM), Dublin, Ireland (October 2006).

[4] López de Vergara, A. Guerrero, V.A. Villagrá, 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.

[5] J. Strassner. "Policy-based Network Management: Solutions for the Next Generation". Morgan-Kaufman Publishers. ISBN 1-55-859-1, Sep 2003.

[6] http://www.w3.org/Submission/SWRL/

[7] Dunlop N., Indulska J. Raymond K. A., "Methods for Conflict Resolution in Policy-Based Management Systems", in the Proceedings of the 7th International Conference on Enterprise Distributed Object Computing (EDOC 2003), Brisbane, Australia, 2003.

[8] Dunlop N., Indulska J., Raymond K. A., "Dynamic Policy Model for Large Evolving Enterprises", Proceedings of the Fifth International Conference on Enterprise Distributed Object Computing (EDOC 2001), Seattle, Washington, USA, September, 2001.

[9] Dunlop N., "Dynamic Policy-Based Management in Open Distributed Environments", Ph.D. Thesis, University of Queensland, Brisbane, Australia, September, 2002.

[10] Dunlop N., Indulska J., Raymond K. A., "A Formal Specification of Conflicts in Dynamic Policy-Based Management Systems", DSTC Technical Report, CRC for Enterprise Distributed Systems, University of Queensland, August, 2001.

[11] Dunlop N., Indulska J., Raymond K. A., "Dynamic Conflict Detection for Large Evolving Enterprises", Proceedings of the Sixth International Conference on Enterprise Distributed Object Computing (EDOC 2002), Lausanne, Switzerland, September, 2002.

[12] Charalambides M., Pavlou G., et. al, "Policy Conflict Analysis for DiffServ Quality of Service Management", IEEE Transactions on Network and Service Management, vol.6, no.1, March 2009.

[13] Kamoda H., Yamaoka M., Matsuda S., Broda K. and Sloman M. 2005, "Policy conflict analysis using free variable tableaux for access control in web services environments", in Proceedings of the Policy Management for the Web Workshop at the 14th International World Wide Web Conference (WWW).

[14] SLA Management Handbook – Concepts and Principles, Release 2.5, Telemanagement Forum, July 2005.

[15] Yan Lindsay Sun, Wei Yu, Zhu Han, Liu, K.J.R., Information Theoretic Framework of Trust Modeling and Evaluation for Ad Hoc Networks, Journal on Selected Areas in Communications, IEEE, Feb. 2006, Volume: 24 Issue:2, p: 305 – 317.

[16] T. M. Cover and J. A. Thomas, Elements of Information Theory. New York: Wiley, 1991.

[17] L. Mamatas, S. Clayman, M. Charalambides, A. Galis and G. Pavlou, "Towards an Information Management Overlay for Emerging Networks", 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010), 19-23 April 2010, Osaka, Japan.

[18] Y. Myoung Ko, Epidemic-Based Information Dissemination in Wireless Mobile Sensor Networks, IEEE/ACM Transactions on Networking, Vol. 18, No. 6, December 2010.

[19] G. Gehlen, F. Aijaz, M. Sajjad, B. Walke, A Rule Based Publish/Subscribe Context Dissemination Middleware, IEEE WCNC, Wireless Communications and Networking Conference, 2007.

[20] C. Anagnostopoulos, S. Hadjiefthymiades and E. Zervas, Information Dissemination between Mobile Nodes for Collaborative Context Awareness, IEEE Transactions on Mobile Computing, Vol.10, No.12, December 2011.

[21] C. Intanagonwiwat, R. Govindan and D. Estrin, Directed diffusion: A scalable and robust communication paradigm for sensor networks, in Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00), August 2000, Boston, Massachussetts.

[22] G. Gehlen, F. Aijaz, M. Sajjad, B. Walke, A Rule Based Publish/Subscribe Context Dissemination Middleware, IEEE WCNC, Wireless Communications and Networking Conference, 2007.

[23] F. S. Correa da Silva, et al, On the insufficiency of ontologies: problems in knowledge sharing and alternative solutions, Elsevier, Knowledge-Based Systems 15 (2002) 147-167.,

[24] R. G. Clegg, S. Clayman, G. Pavlou, L. Mamatas and A. Galis, "On the selection of management/monitoring nodes in highly dynamic networks", IEEE Transactions on Computing, 2012, to appear.

[25] S. Clayman, R. G. Clegg, L. Mamatas, G. Pavlou and A. Galis, "Monitoring, Aggregation and Filtering for Efficient Management of Virtual Networks", 7th International Conference on Network and Service Management CNSM 2011.,

[26] S. Clayman, A. Galis, L. Mamatas, "Monitoring Virtual Networks with Lattice", 12th IEEE/IFIP NOMS 2010 - International Workshop on Management of the Future Internet (ManFI 2010), Osaka, Japan, 2010.,

[27] L. Mamatas, S. Clayman, M. Charalambides, A. Galis and G. Pavlou, "Towards an Information Management Overlay for Emerging Networks", 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010), 19-23 April 2010, Osaka, Japan.

[28] http://wiki.univerself-project.eu/files/2614-3229

[29] M.Wadczak, T.B. Meriem, B.Radier, R.Chaparadza, K.Quinn, J. Kielthy, B.Lee, L.Ciavaglia, K.Tsagkaris, S.Szott, A.Zafeiropoulos, A.Liakopoulos, A.Kousaridas, M.Duault, "Standardizing a reference model and autonomic network architectures for the self-managing future internet", IEEE Network, vol.25, 2011, pp. 50-56

[30] TeleManagement Forum (TMForum), http://www.tmforum.org/browse.aspx

[31] 3rd Generation Partnership Project (3GPP): http://www.3gpp.org/SON

[32] [Next Generation Converged Operations Requirements Phase 1", A Deliverable by the NGMN Alliance, available at http://www.ngmn.org/uploads/media/NGCOR_Phase_1_Final_Deliverable.pdf

[33] http://www.itu.int/rec/T-REC-Y/en

[34] IETF Internet Draft: A Framework for Autonomic Networking, draft-behringer-autonomic-network-framework-00 available at: https://datatracker.ietf.org/doc/draft-behringer-autonomic-network-framework/

10 Abbreviations

3GPP	3rd Generation Partnership Project
3GPP LTE	3GPP Long Term Evolution
3GPP SAE	3GPP Service Architecture Evolution
AFI	Autonomic network engineering for the self-managing Future Internet
AP	Access Point
API	Application Programming Interface
BoF	Birds-of-a-Feather
BSS	Business Support System
CAPEX	Capital Expenditures
DiffServ	Differentiated services
DoW	Description of Work
E2E	End-to-End
EMS	Element Management System
eNodeB	Evolved NodeB
ETSI	European Telecommunications Standards Institute
FG-FN	Focus Group – Future Networks
FMC	Fix Mobile Convergence
FTTH	Fibre To The Home
GUI	Graphical User Interface
GW	Gateway
H2N	Human-to-Network
ICT	Information and Communication Technologies
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IRTF	Internet Research Task Force
IMS	IP Multimedia Subsystem
IP	Internet Protocol
IRTF	Internet Research Task Force
IS	Information System
IT	Information Technology
ITU	International Telecommunication Union
ITU-T	International Telecommunication Union – Telecommunications standardization sector
KPI	Key Performance Indicator
LCCN	Learning-Capable Communication Networks
LE	Large Enterprises
LSP	Label Switched Path
LTE	Long Term Evolution
LTE-A	LTE Advanced
MPLS	Multi Protocol Label Switching
NaaS	Network as a Service
NMRG	Network Management Research Group
NMS	Network Management System
OAM	Operations Administration and Maintenance
OFDM	Orthogonal Frequency-division Multiplexing
OFDMA	Orthogonal Frequency-Division Multiple Access
OPEX	Operational Expenditures

OSS	Operations Support System
PDN-GW	Packet Data Network Gateway
QoE	Quality of Experience
QoS	Quality of Service
ROI	Return of Investment
RAN	Radio Access Network
RRM	Radio Resource Management
SGW	Serving Gateway
SME	Small and Medium Enterprises
SLA	Service Level Agreement
SON	Self Organized Networks
TCO	Total Cost of Ownership
TMF	TeleManagement Forum
UC	Use case
UMF	Unified Management Framework
VoIP	VoIP - Voice over IP
VPN	Virtual Private Network

11 Definitions

Atomic NEM – *NEM the internal functioning of which relies only on one equipment.*

Composite NEM – *NEM the internal functioning of which can rely on separated piece of software running on different equipments.*

Coordination block (COORD) – *A core UMF block that aims to ensure the proper sequence in triggering of NEMs and the conditions under which they will be invoked (i.e. produce their output), taking into account operator service and scenario requirements and at the same time the needs for conflict avoidance, stability control and joint optimization through the corresponding functions.*

Functional requirement – I*t is a description of a function, or a feature of a system, or its components, capable of solving a certain problem or replying to a certain need/request. The set of functional requirements present a complete description of how a specific system will function, capturing every aspect of how it should work before it is built, including information handling, computation handling, storage handling and connectivity handling.*

Governance block (GOV) – *A core UMF block that aims to give a human operator a mechanism for controlling the network from a high level business point of view, that is, without the need of having deep technical knowledge of the network.*

Knowledge block (KNOW) – *An infrastructure that uses and/or manipulates information and knowledge, including information/knowledge flow optimization within the network.*

Network Empowerment Mechanism (NEM) – *A functional grouping of objective(s), context and 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 network to enhance or simplify its control and management (e.g. take over some operations). An intrinsic capability of a NEM is to be deployable and interoperable in a UMF context (in a UMF-compliant network).*

NEM class – *it is a piece of software that contains the logic achieving a specific autonomic function. Such class is deployed in a network running a UMF system and requires being instantiated on a set of concrete network elements to effectively perform its autonomic function*

NEM (class) instance – *it allows performing a given autonomic function onto a given sub-set of a network. This is achieved by binding the code of a NEM class to a set of identified network resources/equipments. This NEM instance is identified by an instance ID and its unique interface with the UMF. This NEM instance at any given time is handling a set of identified network resources (this set can evolve with time). Hence there may be multiple instances of a given NEM class inside the same network e.g. one per area). A NEM instance is created by the UMF system it is being deployed in.*

NEM instance description – *it describes a given instance of a given NEM class. This description is issued by the NEM instance towards UMF system, it is used for the registration of the NEM and it tells which information is monitored and which actions are taken.*

NEM instance description grammar – *it is a subset of UMF specifications describing which information MUST and MAY be provided by the NEM instance when starting (and when its settings are changed) so as to register to the UMF system the: a) capabilities of this NEM instance regarding information/knowledge sharing, b) requirements of this NEM instance regarding knowledge inputs and c) conflicts of this NEM instance with already running NEM instances of any NEM class.*

NEM mandate – *it is issued by the UMF system to a NEM instance. This NEM Mandate is a set of instructions telling which equipments MUST be handled by this NEM instance and which settings this NEM instance MUST work with.*

NEM mandate format – *it is a subset of UMF specifications describing which information MUST and MAY be provided by the UMF system to the NEM.*

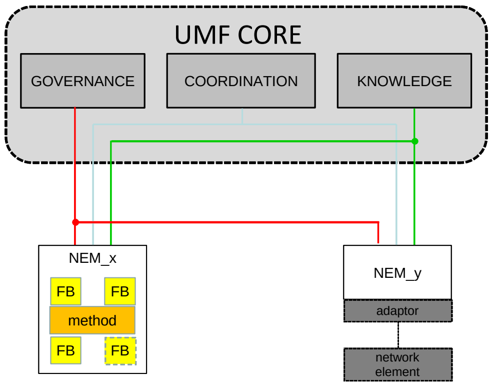
NEM manifest – *it describes a given NEM class. This description provides guidance to the network operator in order to install and configure an instance of this NEM class – the goal of a NEM manifest is similar to a datasheet). This description is issued by the NEM designer towards network operators.*

NEM manifest grammar – *it is a subset of UMF specifications describing which information MUST and MAY be provided by the NEM developers in order to describe their NEM class.*

NEM skin – *Software component common to all NEMs. It provides to the NEM developer the UMF interfaces and the de-facto NEM behaviour (i.e. registration, configuration, knowledge-exchange and management) needed for interaction with the UMF core and compliance with the UMF specs.*

NEM specifications – *they constrain the behaviour of NEMs and define the generic part of their interfaces with UMF elements.*

Unified Management Framework (UMF) – *A framework that will help produce the unification, governance, and "plug and play" of autonomic networking solutions within existing and future management ecosystems. The objective of the UMF is to facilitate the seamless and trustworthy deployment of NEMs. The UMF has three core blocks that are used by the NEMs to achieve this, as shown in the figure below.*



Use case – *A descriptor of a set of precise problems to be solved. It describes steps and actions between stakeholders and/or actors and a system, which leads the user towards an added value or a useful goal. A use case describes what the system shall do for the actor and/or stakeholder to achieve a particular goal. Use-cases are a system modelling technique that helps developers determine which features to implement and how to gracefully resolve errors.*

12 Annex A: Restful UMF API description

In order to support the UMF specifications in practice, an Application Programming Interface (API) in the form of a library redistributable has been developed, namely the "UMFCommon". The purpose of such a library is twofold: provide an abstraction layer of UMF interactions and RESTful details to the NEM developer while complying with the specifications so that any other UMF-compliant implementation, disregarding of the hosting platform or the programming language, will be able to use the RESTful interfaces exposed by UMFCommon. So, although this library has been implemented in JAVA as a proof of concept, the published RESTful interfaces can be used by any web technology and vice versa, i.e., the library can be exploited in such a way that it utilizes the same information model with a communication technology other than HTTP.

The largest part of UMFCommon consists of the so called "NEM skin". As one would expect, the skin is the common behaviour between all NEMs including their interfaces to CORE blocks (management and knowledge-exchange). For the seamless binding of JAVA methods to HTTP resources, an underlying mechanism has been implemented and included in the package. In practice, the final result, is that one developer might build a UMF compliant NEM using UMFCommon and without having to deal with any particular UMF workflow, specification or HTTP detail, while the other end (e.g. CORE) might be implemented in any web-based, RESTfull client or server technology.

13 Annex B: Data Dictionary

13.1 Overview of SID Policy Model

This annex presents an overview of the SID Policy Model [Shared Information/Data (SID) Model. Addendum 1-POL Common Business Entity Definitions – Policy. GB922 Addendum 1- POL. Version 1.3]. The goal is to identify how the SID model can be used inside UMF, and identify possible needs for extensions.

SID organizes the Policy Domain in 4 collections of business units, called Aggregated Business Units (ABEs): Policy, Policy Specification, Policy Application and Policy Management, as depicted in Figure 41.

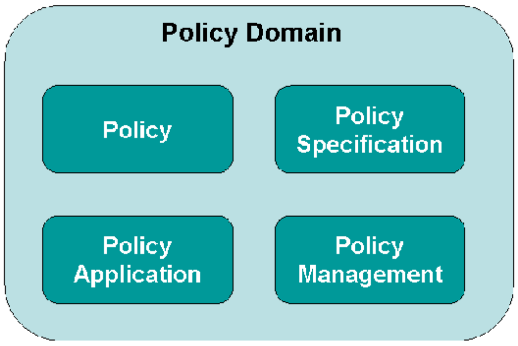


Figure 41. SID Policy Domain.

The Policy ABE defines the core groups of entities that are used to represent policy, independent of its content. The Policy Specification ABE templatizes these five entities. The Policy Application ABE defines how policy applications can be built, and the Policy Management ABE associates policy entities with other SID entities. These are shown in Figure 42below, where the refinement into the second level ABEs is presented:.

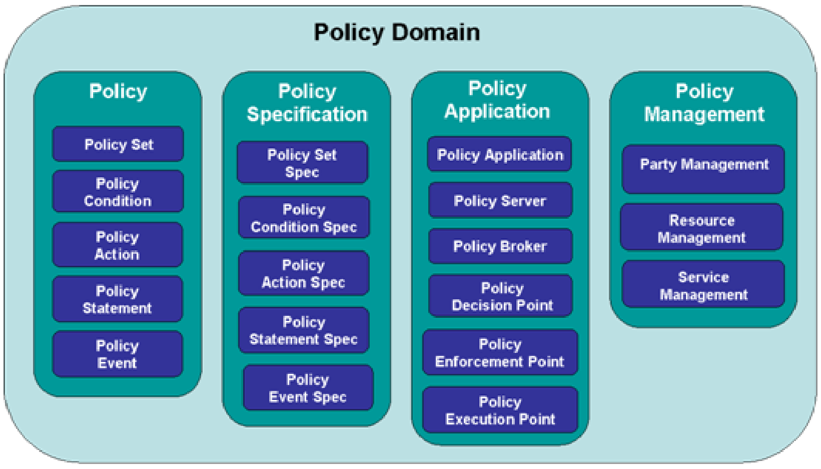


Figure 42. Level Two of the Policy Domain of the SID Framework.

13.1.1 Policy

The main entity in the policy domain is the PolicyRule, defined as an intelligent data container. It contains data that define how the PolicyRule is used in a managed environment as well as a specification of behaviour that dictates how the managed entities that it applies to will interact. The contained data is of four types: (1) data and metadata that define the semantics and behaviour of the policy rule and the behaviour that it imposes on the rest of the system, (2) a group of events that can be used to trigger the evaluation of the condition clause of a policy rule, (3) a group of conditions aggregated by the PolicyRule, and (4) a group of actions aggregated by the PolicyRule.

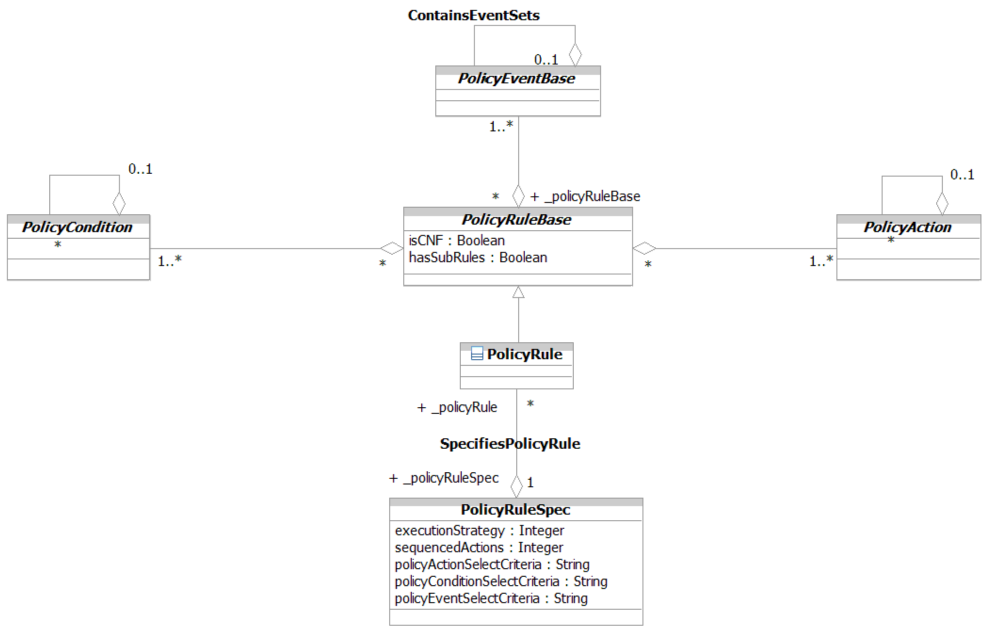


Figure 43. Representation of a PolicyRule.

PolicyRules are built from PolicyRuleSpecifications (called PolicyRuleSpec in the model). A PolicyRuleSpec acts as a mechanism to specify the invariant (i.e., non-changeable) features and behavior that makes up a Policy. A PolicyRuleSpec has two important attributes that all PolicyRules have, called executionStrategy and sequencedActions.

The executionStrategy attribute is an enumerated integer that defines the strategy to be used when executing the sequenced actions aggregated by this PolicyRule. Defined execution strategies include:

1. Do Until Success
2. Do All
3. Do Until Failure
4. Do All Without Failure or Do Nothing

The sequencedActions attribute is an enumerated integer that defines how the ordering of the PolicyActions associated with this PolicyRule is to be interpreted. Values include:

1. Mandatory
2. Recommended
3. Best Effort

The PolicyRule entity itself defines two attributes, isCNF and hasSubRules. PolicyConditions can be represented in two different forms, called Conjunctive Normal Form (an AND of ORs) and Disjunctive Normal Form (an OR of ANDs). The isCNF attribute defines which one of these forms the PolicyCondition clause is.

A PolicyRule is designed to be used for a single purpose. Sometimes, a management system needs multiple separate policy decisions and actions to be conducted in concert. A PolicyGroup is a generalized aggregation container. It enables PolicyRules and/or PolicyGroups to be aggregated in a single container

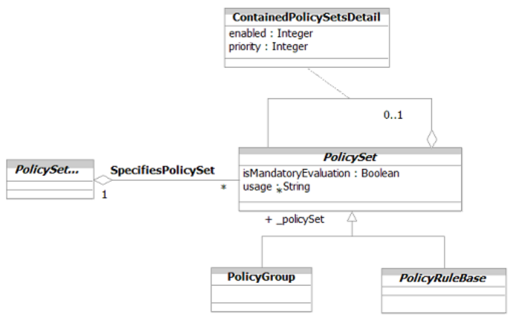


Figure 44. Policy Set.

Both a PolicyGroup as well as a PolicyRule can act as intelligent containers. This common capability is abstracted and generalized into a common superclass, called PolicySet. A PolicySet can therefore be usedto define common semantics for PolicyRulesand PolicyGroups.

The ContainedPolicySets aggregation is used to gather together discrete PolicySet objects to form a group of PolicySet objects. Such a group must share the same DecisionStrategy. Its semantics are implemented by the ContainedPolicySetsDetail association class.

The ContainedPolicySetsDetail association class represents the semantics of the ContainedPolicySets aggregation. It provides additional semantics that enable this grouping of PolicySets to be prioritized and enabled, so that they can interwork with other PolicyRules and PolicyGroups.

PolicyEvents are significant occurrences that trigger the evaluation of one or more PolicyRules. The composite pattern is used to define atomic and composite PolicyEvents:

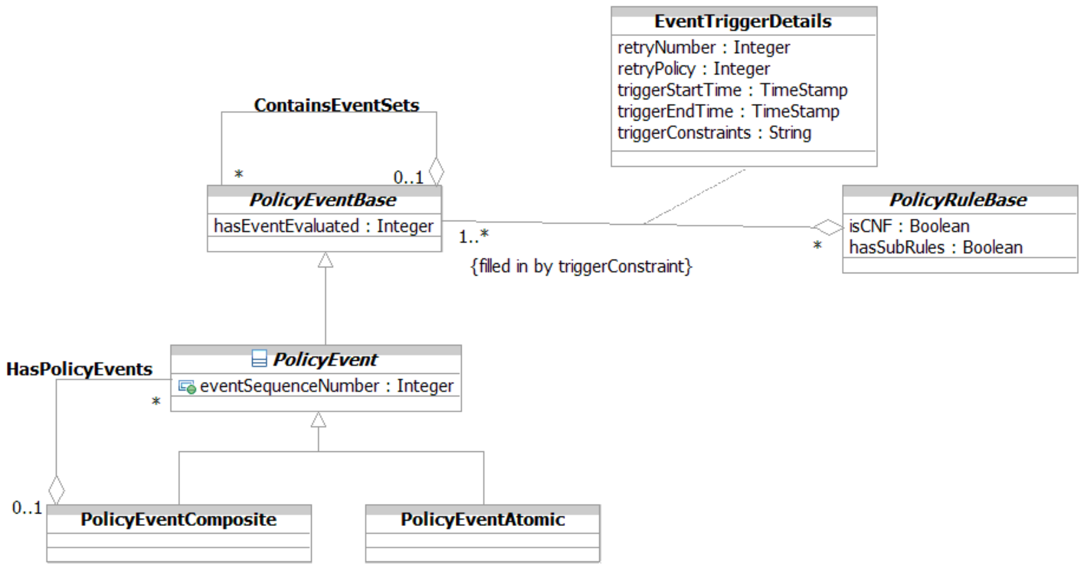


Figure 45. Policy Events and Policy Sets.

The composite pattern is used to build atomic and composite PolicyEvents. A PolicyEventAtomic is a base class that represents the occurrence of a single atomic event, which is used to trigger the evaluation of the condition clause of a PolicyRule. In contrast, a PolicyEventComposite is a base class that represents the occurrence of a composite event. A composite event is an event that is made up of a set of PolicyEventAtomic and/or

PolicyEventComposite entities. Like a PolicyEventAtomic, a PolicyEventComposite can also be used to trigger the evaluation of the condition clause of a PolicyRule.

There is significant similarity between the most common types of PolicyConditions and PolicyActions. Specifically:

PolicyConditions are of the form: “IF <policy-condition> is TRUE”

PolicyActions are of the form: “SET <action-target> to <value>

Both the condition clause and the action clause are in reality of the same form:

{ variable, operator, value }

where the braces are used to denote a tuple. To see this, we can write a PolicyCondition as:

IF <variable><operator><value> is TRUE

This enables the model to generalize the standard form of a PolicyCondition and a PolicyAction into an object that is called a PolicyStatement. This has important implications, since a Policy Decision Point, a Policy Enforcement Point, and a Policy Execution Point can now share the same basic syntax. Both PolicyConditions as well as PolicyActions share the same variables and values; the difference in semantics is reflected in the types of operators that are allowed to be used for PolicyConditions versus PolicyActions

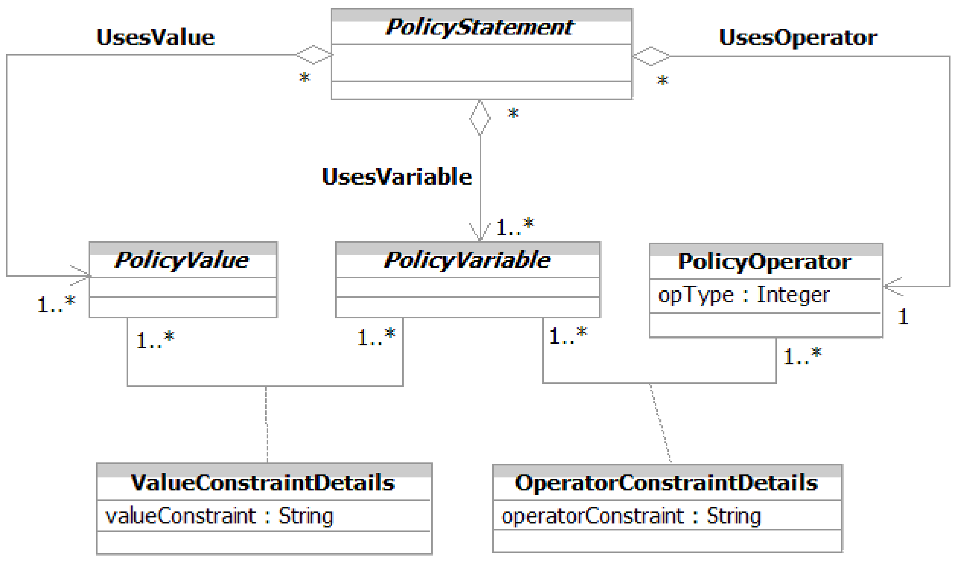


Figure 46. Policy Statement.

The condition clause of a PolicyRule is represented by a Policycondition. This class can be used to represent rule-specific or reusable policy conditions

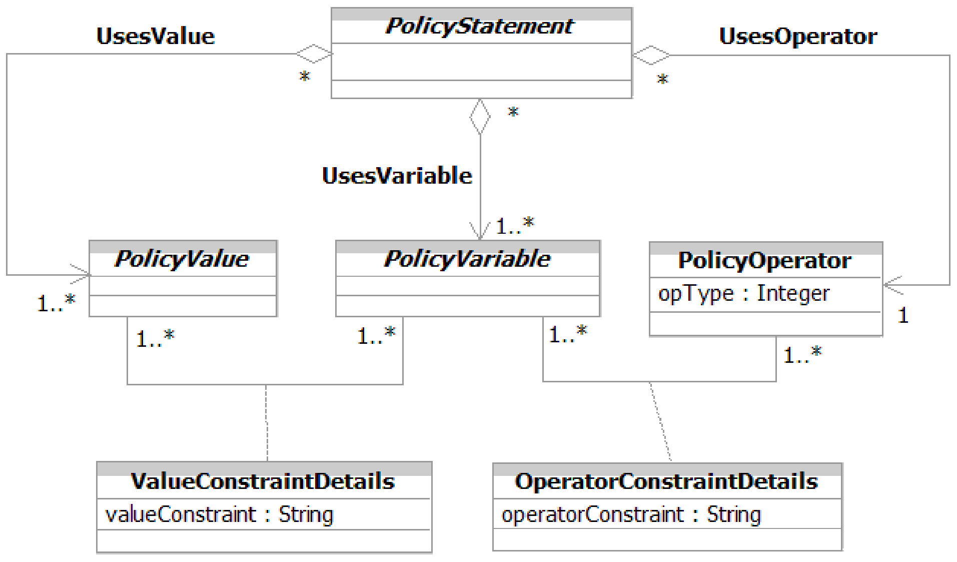


Figure 47. Policy Condition.

PolicyAction is an abstract base class that represents how to form the action clause of a PolicyRule. This consists of a single occurrence of a PolicyStatement, which is of the form: {variable, operator, value}.

Policy actions have the semantics of "SET variable to value", which is implemented by a PolicyStatement. In order to provide flexibility, DEN-ng defines two types of actions:

- pass actions are invoked if the condition clause was TRUE
- fail actions are invoked if the condition clause was FALSE

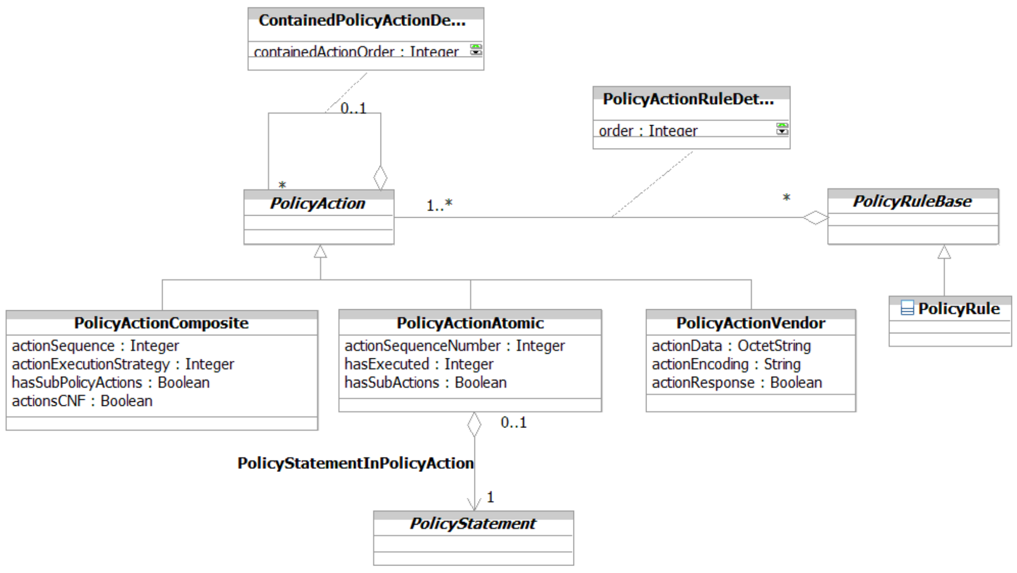


Figure 48. Policy Action.

13.1.2 Policy Application ABE

A **PolicyApplication** is a special type of entity for use in policy-based management applications, and it is used for defining relationships to different managed entities. It has four principal subclasses: PolicyServer, PolicyDecisionPoint (PDP), PolicyExecutionPoint (PXP) and PolicyEnforcementPoint (PEP). This relationship is shown in Figure 49.

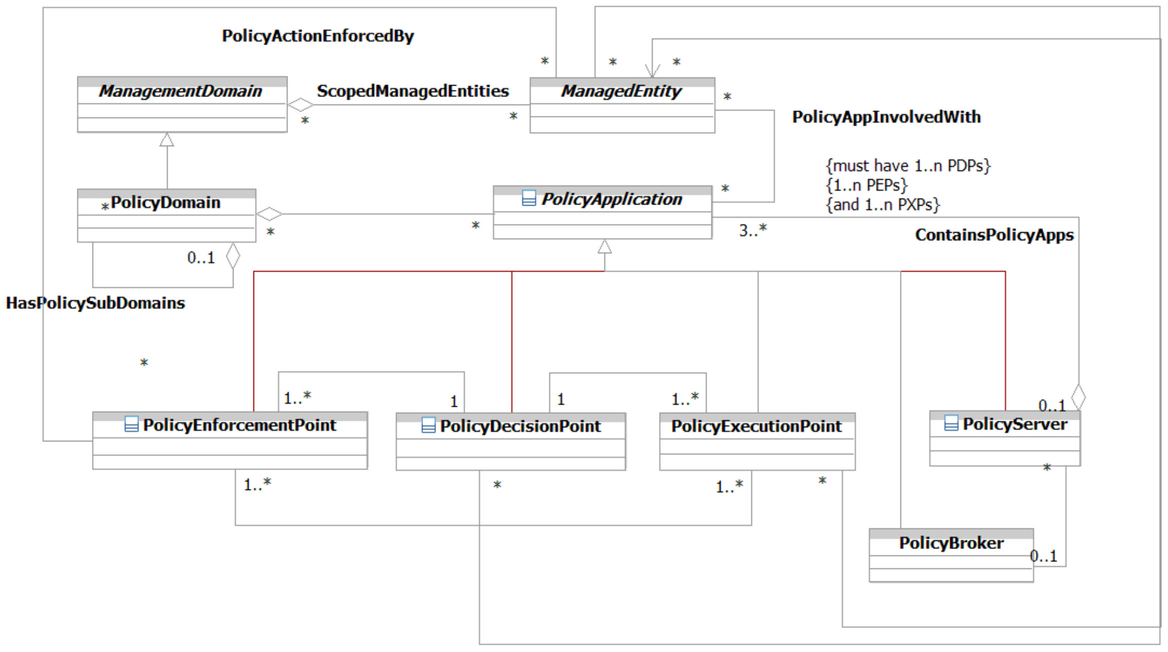


Figure 49. Policy Application simplified view.

A **PolicyServer** is a fundamental building block of a policy-based management system. It represents both a set of core functionality for implementing policy as well as a unit of distribution in a distributed implementation. A PolicyServer is an entity that is either manufactured or is constructed by integrating the functionality of different PDPs, PXPs, PEPs, and other applications. These other applications provide the logic to manage and control the set of PDPs, PXPs, and PEPs that constitute a PolicyServer.

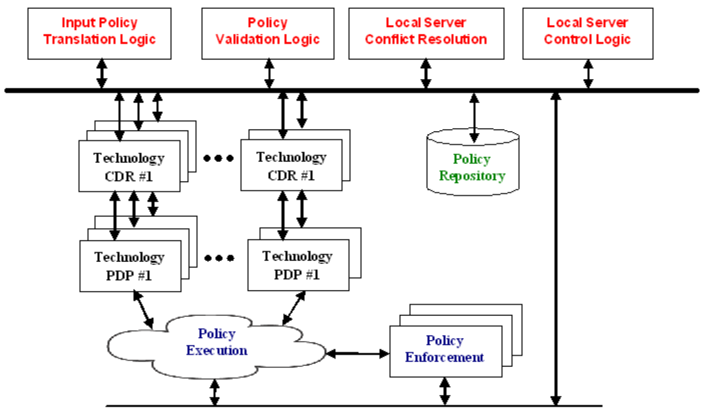


Figure 50. Anatomy of a Policy Server.

PolicyServers affect ManagedEntities in a particular PolicyDomain, and are coordinated through a **PolicyBroker**. The purpose of the PolicyBroker is to control how different PolicyServers interact with each other. In this regard, it has two different functions.The first function is to ensure that conflicts between different policy rules don't exist when different Policy Servers are asked to work together. The second is to coordinate the application of different policies in different Policy Servers.

A **PolicyDecisionPoint** makes policy decisions for itself or for other entities that request such decisions, such as PolicyEnforcementPoints (PEPs) and PolicyExecutionPoints (PXPs). PDPs use policies to configure or answer queries from policy-capable network elements or from an operator. One or more PolicyDecisionPoints are contained in a PolicyServer.

A **PolicyExecutionPoint** is an entity that executes a policy decision given to it by a PDP. A **PolicyEnforcementPoint** is is used to verify that a prescribed set of PolicyActions have been successfully executed on a set of PolicyTargets. A PolicyEnforcementPoint serves as an interface between the devices that policy is executed on and the policy decision-makers (such as the PolicyDecisionPoint) of the policy. PolicyEnforcementPoints request work to be performed from PolicyDecisionPoints, and then enforce decisions made by PolicyExecutionPoints on their PolicyTargets.

13.1.3 Policy Management ABE

This ABE describes how policy can be used to manage different types of managed entities.

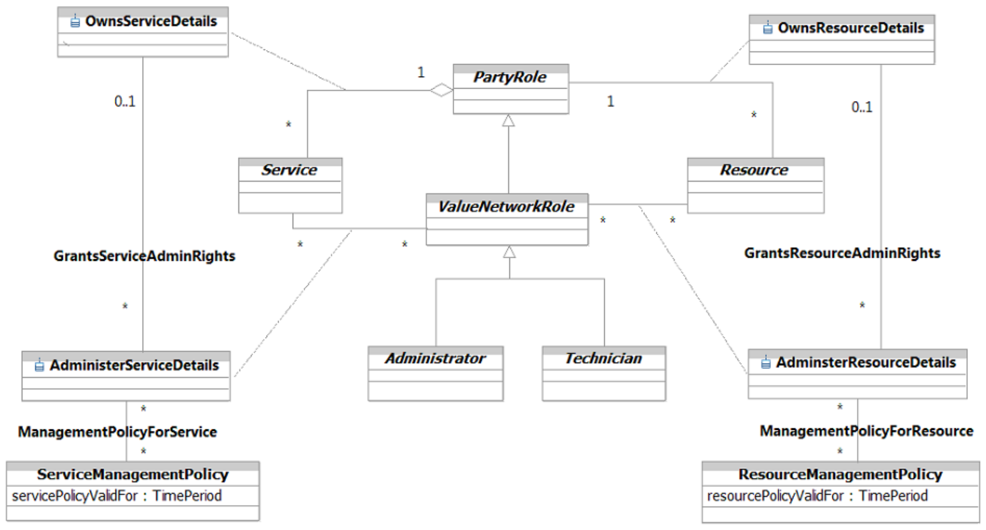


Figure 51. Using Policy and PartyRoles to Manage Resources and Services.

A **PartyRole** defines the function that a Party takes on. PartyRoles can represent the ability to manage, configure, use, and perform other types of interactions with LogicalResources. Service and Resource management methods are symmetrical.

The **OwnsResource**association defines the set of Resources (PhysicalResources and/or LogicalResources) that a particular PartyRole owns. The **AdministersResource** association defines the set of Resources (PhysicalResources and/or LogicalResources) that a particular Party, playing the role of **ValueNetworkRole**, administers. From a business perspective, the Owner has to either appoint or give permission to the Administrator to administer the Resource that is owned. This is done using the **GrantsResourceAdminRights** association.

The **ResourceManagementPolicy** class defines the particular policies that are used to define how different aspects of the Resource are managed and maintained.

13.1.4 Policy Specification ABE

The main purpose of all entities in the PolicySpecification ABE, from the business point-of-view, is to provide a standardized structure of the entity that the Spec refers to. Policy templates provide a very important advantage – the ability to define standard templates, or specifications, that can be used to manage different types of managed entities. Reader may refer to Policy Addendum in order to get deeper description of the Policy Specification ABE.

13.2 UMF info model diagrams

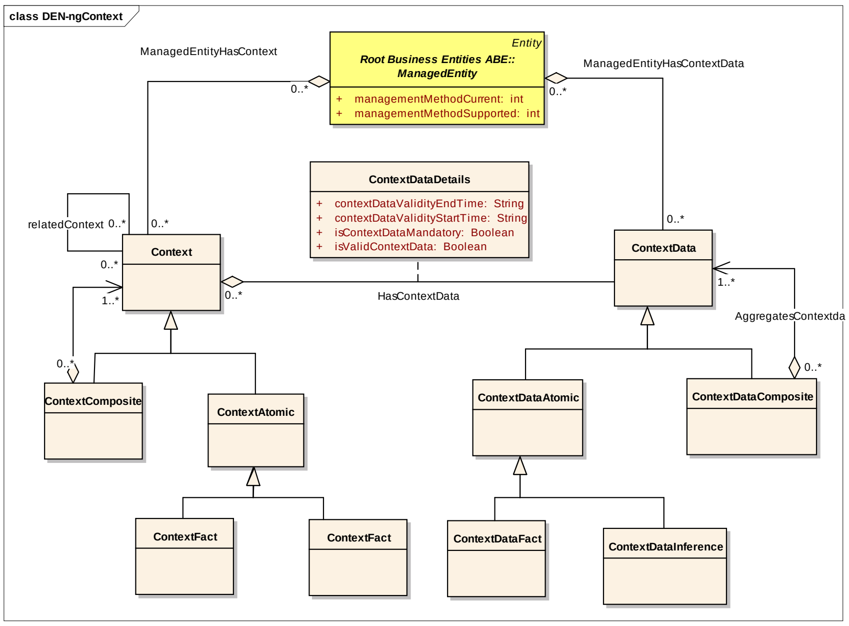


Figure 52. Information model of DEN-ng Context

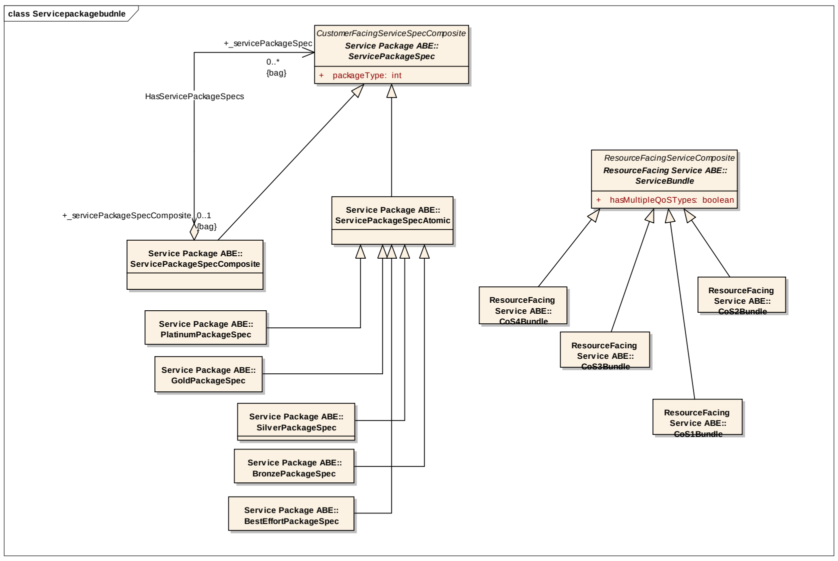
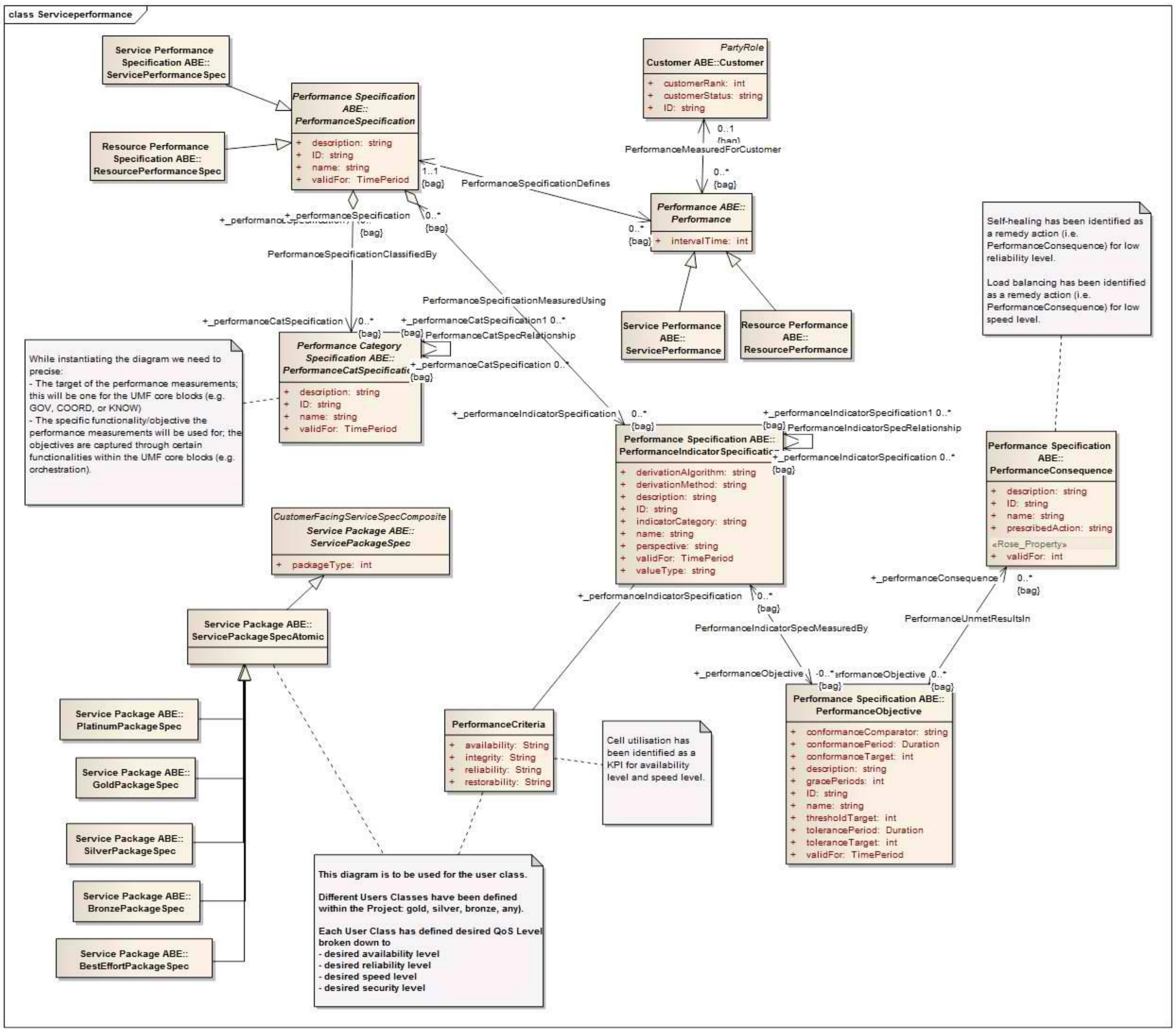


Figure 53. Information model of Service package bundle


class Serviceperformance
Service Performance Specification ABE:: ServicePerformanceSpec
Resource Performance Specification ABE:: ResourcePerformanceSpec
Performance Specification ABE:: PerformanceSpecification
+ description: string
+ ID: string
+ name: string
+ validFor: TimePeriod
1..1 {bag}
PerformanceSpecificationDefines
+_performanceSpecification
0..* {bag}
PerformanceSpecificationClassifiedBy
PerformanceSpecificationMeasuredUsing
+_performanceCatSpecification
+_performanceCatSpecification1 0..*
PerformanceCatSpecRelationship
Performance Category Specification ABE:: PerformanceCatSpecificatio
+ description: string
+ ID: string
+ name: string
+ validFor: TimePeriod
PartyRole
Customer ABE::Customer
+ customerRank: int
+ customerStatus: string
+ ID: string
0..1 {bag}
PerformanceMeasuredForCustomer
Performance ABE:: Performance
+ intervalTime: int
Service Performance ABE:: ServicePerformance
Resource Performance ABE:: ResourcePerformance
+_performanceIndicatorSpecification
+_performanceIndicatorSpecification1 0..*
PerformanceIndicatorSpecRelationship
Performance Specification ABE:: PerformanceIndicatorSpecificatio
+ derivationAlgorithm: string
+ derivationMethod: string
+ description: string
+ ID: string
+ indicatorCategory: string
+ name: string
+ perspective: string
+ validFor: TimePeriod
+ valueType: string
Performance Specification ABE:: PerformanceConsequence
+ description: string
+ ID: string
+ name: string
+ prescribedAction: string
«Rose_Property»
+ validFor: int
+_performanceConsequence
PerformanceUnmetResultsIn
PerformanceIndicatorSpecMeasuredBy
+_performanceObjective
Performance Specification ABE:: PerformanceObjective
+ conformanceComparator: string
+ conformancePeriod: Duration
+ conformanceTarget: int
+ description: string
+ gracePeriods: int
+ ID: string
+ name: string
+ thresholdTarget: int
+ tolerancePeriod: Duration
+ toleranceTarget: int
+ validFor: TimePeriod
CustomerFacingServiceSpecComposite
Service Package ABE:: ServicePackageSpec
+ packageType: int
Service Package ABE:: ServicePackageSpecAtomic
Service Package ABE:: PlatinumPackageSpec
Service Package ABE:: GoldPackageSpec
Service Package ABE:: SilverPackageSpec
Service Package ABE:: BronzePackageSpec
Service Package ABE:: BestEffortPackageSpec
PerformanceCriteria
+ availability: String
+ integrity: String
+ reliability: String
+ restorability: String

While instantiating the diagram we need to precise:
- The target of the performance measurements; this will be one for the UMF core blocks (e.g. GOV, COORD, or KNOW)
- The specific functionality/objective the performance measurements will be used for; the objectives are captured through certain functionalities within the UMF core blocks (e.g. orchestration).

Self-healing has been identified as a remedy action (i.e. PerformanceConsequence) for low reliability level.

Load balancing has been identified as a remedy action (i.e. PerformanceConsequence) for low speed level.

Cell utilisation has been identified as a KPI for availability level and speed level.

This diagram is to be used for the user class.

Different Users Classes have been defined within the Project: gold, silver, bronze, any).

Each User Class has defined desired QoS Level broken down to
- desired availability level
- desired reliability level
- desired speed level
- desired security level

Figure 54. Information model of Service Performance

13.3 Information flow in the UMF

This section aims at providing an overall view of the information which is being exchanged among the UMF core blocks as well as between the UMF core blocks and the NEMs.

Table 14 Information flow in the UMF

Information Concepts and Attributes	Interface	Clarifications - Examples
Operator High Level Objectives (HLOs), associated to • Network Operation, • User Class, • Targeted QoS Level, • Newly-deployed Service.	NO-GOV	The network operations are related to NO objectives. User class(es) are linked to NO objectives.

		The QoS level is linked to NO objectives.
Service Attributes • Service Type (streaming, interactive, background, p2p, IP-TV, Internet) • Network Technology (Wi-Fi, LTE, FTTH) • Supported QoS Level ○ Supported Availability Level ○ Supported Reliability Level ○ Supported Speed Level ○ Supported Security level (excellent, normal, critical) ○ Associated QoS cost • Service Requirements ○ required availability level ○ required reliability level ○ required speed level ○ required security level ○ associated QoS cost	NO-GOV	The Service Attributes are communicated between the NO and the GOV as well between the COORD and the GOV. The Supported QoS Level refers to the different values of the parameters which are supported by the Service and can be provided to the user. The Service Requirements capture the typical values according for example to the network technology. QoS cost is for the specific service deployment and desired QoS level.
Network Information • Network configuration ○ Configuration constraints ○ Network parameter configuration ○ Network Topology ○ Network status • Resource Configuration (e.g. Router,…) • Network alert (cell outage, other?) • Network Monitoring Information (from NEMs) ○ Monitoring requirements ▪ Required type of monitoring information • Example: link usage for a set of routers ▪ Required type of monitoring frequency • Example: every 5 sec min ○ Is aggregation needed? ○ Is aggregated? • Network Performance Measurements ○ Target ○ Type (e.g. joint optimization performance, orchestration performance, …) ○ Metric []	NEM-KNOW KNOW-COORD KNOW-GOV	This refers to low –level network information as provided to the KNOW block by the NEMs. Network measurements are related to low-level network information, so it requires monitoring facilities within the NEMs. A summary of the network measurements could be stored in the KNOW block.
Policy • RAT Policy • Policy Argument • Service Policy • Network Policy • User Policy • NEM Policy • COORD Policy ○ weights for aggregation of utilities/utility functions • KNOW Policy ○ Optimisation Goal ▪ optGoalId	NEM-GOV GOV-COORD GOV-KNOW	RAT Policy: policy applied to RAT Policy argument is super class of RAT, Network, Service etc Service policy for service provision Business level Policy Service level Policy Commands: derived by NEMs based on NEMs

▪ optGoalParameters ▪ optGoalStatus ○ AccuracyObjectives ▪ objectiveId ▪ objectiveParameter ▪ objectiveStatus ▪ is filtering used		policies. Can be overridden by Control Policy, i.e. configuration not derived by NEMs but enforced to NEMs and resources. Opt. Goal- example: “Optimize everything in order to save energy”. The KNOW functions adapt their tactics in order to meet this goal. For example, they reduce communication overhead using information filtering etc. Accuracy objective: For information filtering. “Please pass me the value whenever it changes more than 5%. I don’t mind having less accuracy in order to save energy (to meet the goal coming from GOV) through reducing communication overhead.
User Class • Type (gold, silver, bronze, any) • User Preferences ○ Desired QoS level ▪ Desired availability level ▪ Desired reliability level ▪ Desired speed level ▪ Desired security level		User Class is Linked to QoS.
NEM • NEM Manifest • NEM Mandate • NEM Information • NEM Instance ○ NEM instance id ○ parameters which are affected by NEM operation (e.g. antenna tilt, managed rerource configuration) ○ metrics affected by NEM Operation (e.g link load, throughput) ○ NEM timing ○ convergence time (method to produce solution) ○ expected interval between two triggers of the NEM ○ NEM utility (related to NEM target) • NEM Configuration ○ NEM Configuration Parameter ▪ Id ▪ Access rights (r/w/a) ▪ Parameter description ○ Current monitoring information ○ Current monitoring frequency • Is NEM information source (for UMF) • NEM text description	COORD-KNOW COORD-NEM COORD-GOV GOV-KNOW NEM-KNOW COORD-GOV NEM-COORD NEM-KNOW	All these concepts and attributes are related to NEM Information Model. In most cases it is expected that the object-level information (e.g. NEM Manifest) will be deduced from the NEM Information Model.

• NEM Conflict (List) ○ Is atomic • NEM Status • NEM monitoring capabilities ○ Available type of monitored information [] ▪ Example: link usage for a set of routers ○ Available frequency of monitoring ▪ Example 5 sec max		

The above presented information is reflected in the following figure (Figure 55. UMF Information flow).

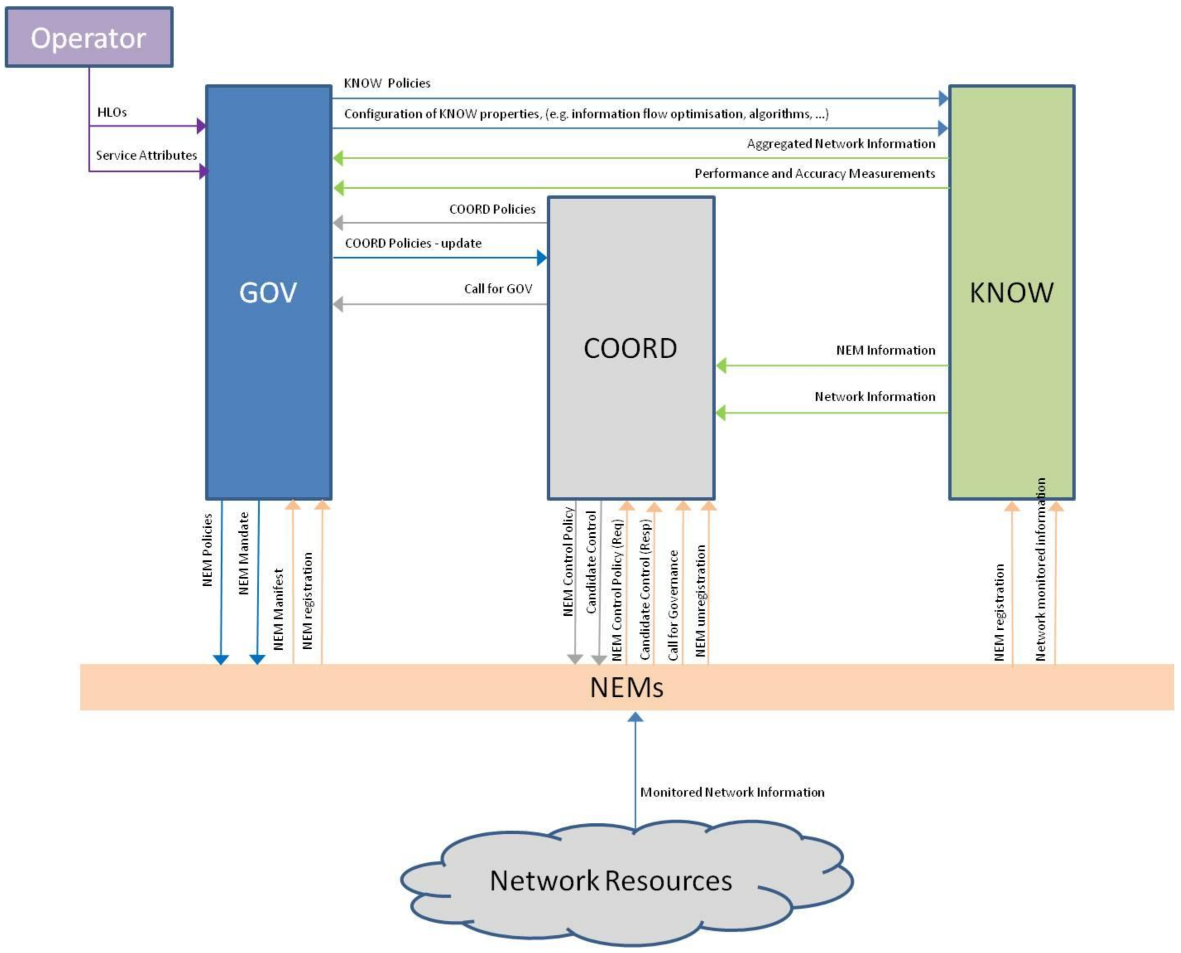


Figure 55. UMF Information flow