Deliverable D2.4

Unified Management Framework (UMF) Specifications Release 3

Grant Agreement	257513	
Date of Annex I	08 October 2013	
Dissemination Level	Public	
Nature	Report	
Work package	WP2 – Unified Management Framework	
Due delivery date	01 June 2013	
Actual delivery date	11 November 2013	
Lead beneficiary	ALBLF	Laurent Ciavaglia (laurent.ciavaglia@alcatel-lucent.com)

SEVENTH FRAMEWORK PROGRAMME

FUTURE NETWORKS

Authors	UPRC – Kostas Tsagkaris, Aristi Galani, Nikos Koutsouris, Panagiotis Demestichas, Aimilia Bantouna, George Poulios, Panagiotis Vlacheas TCF – Gerard Nguengang, Mattieu Bouet ALBLF – Pierre Peloso, Laurent Ciavaglia TID –Beatriz Fuentes UCL –Lefteris Mamatas, Stuart Clayman, Alex Galis UniS –Stylianos Georgoulas, Majid Ghader Fraunhofer – Mikhail Smirnov NEC – Zarrar Youssaf VTT – Teemu Rautio, Marja Liinasuo FT - Zwi Altman, Imen Grida Ben Yahia, Christian Destré NKUA - Evangelos Kosmatos, Konstantinos Chatzikokolakis, Roi Arapoglou, Eleni Patouni, Nancy Alonistioni, Makis Stamatelatos, Alexandros Kaloxylos, George Katsikas, Panagiotis Spapis ALUD - Ingo Karla

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 already 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 D2.4 presents a consolidated and wide-ranging functional specification of the UMF, including the detailed specification of the UMF core functionality and the relevant interfaces, the mechanisms to support the main functions of the core blocks and the life-cycle management of the autonomic management applications – the Network Empowering Mechanisms.

Table of Content

1. Introduction

The Unified Management Framework (UMF), which was 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.

Unified Management Framework (UMF) is 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 interworking of autonomic functions (e.g. Network Empowerment Mechanisms - NEMs). As such, UMF aims also the migration from an ecosystem of separate autonomic functions (AFs) towards a coordinated arrangement of AFs.

UMF as a management framework is based on three main functional blocks namely, Governance, Coordination and Knowledge and the interworking with the autonomic management applications – the Network Empowering Mechanisms (NEMs). The UMF Knowledge block (KNOW) plays the role of information / knowledge collection, aggregation, storage/registry, knowledge production and distribution across all UMF functional components. 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. Its main functionality includes Orchestration, Conflict identification, Optimization and Conflict Avoidance. The role of the Governance block, in response to the management needs of and objectives described by the human network operators, is to supervise and control of the behaviour of the underlying autonomic functionalities (NEMs) and the UMF core blocks. 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. The three blocks together are managing the autonomic functions named Network Empowerment Mechanisms (NEMs).

Deliverable D2.4 "UMF Specifications – Release 3" provides a consolidated wide-ranging functional specification of the UMF, including the detailed specification of the UMF core functionality and the relevant interfaces and the mechanisms to support the main functions of the core blocks.

The document is structured as follow: Chapter 2 presents an analysis of the UMF requirements, which motivated the design of the UMF. Chapter 3 presents the UMF Overview including the specifications of UMF core components and NEMs, regarding their functions and their corresponding operations, as well as the relevant interfaces. Chapter 4 presents functional mechanisms that enable the realization of UMF core functionalities. Chapter 5 presents the UMF positioning and interworking with emerging network frameworks. Concluding remarks are presented in Chapter 6.

2. Analysis of UMF Requirements

Current and future networks are comprised of diverse autonomic network functions. This section presents an analysis that motivates the design of the UMF as a framework for autonomic functions (AF) which:

- Improve capital and operational efficiencies for operators through the use of a common organization, automation and operations of all autonomic functions across the different networks.
- Rapid autonomic functions and service innovation through software-based interworking of AFs in UMF.
- A migration from an ecosystem of separate autonomic functions towards a coordinated arrangement of AFs as represented in the following figure

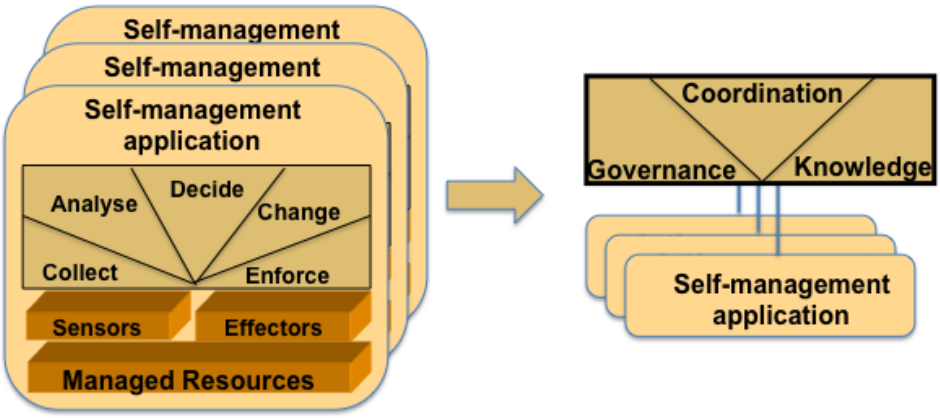


Figure 1 Migration from separate control loops to a coordinated arrangement of multiple control loops

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 2):

- a "bottom-up requirements" expressed through 6 use cases' 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 traditional management architecture and functions (e.g. TMN FCAPS) towards the management functions of Future Networks.

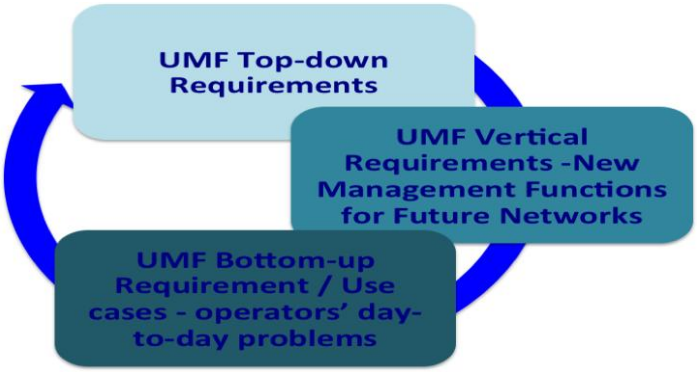


Figure 2. UMF Axes of Requirements

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 deliverables D2.1 and D2.2 – Unified Management Framework, Release 1and 2, and through the analysis work performed in milestones MS26 and MS27 (WP2). The third approach "vertical requirements" aims at elaborating the expected new management functionality of future networks developed so far within deliverable D2.1 and D2.2 – Unified Management Framework, Release 1 and 2 (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 3**).

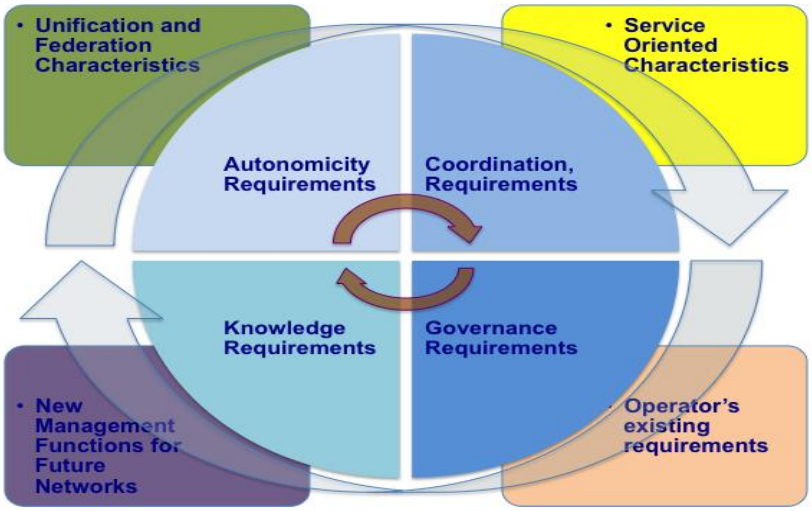


Figure 3. UMF Requirements Synthesis

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 modular and 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 Empowerment Mechanism would encapsulate at least one self-x algorithms/methods and it will be hosted /deployed by the network in case of in-bound management) or by TMN/TMF Network Management Station (NMS) / Operations Support Systems (OSS) in case of out-of-bound management). Such Network Empowerment Mechanisms 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, interworking 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 derived both from the top-down (design goals and principles) and bottom-up (commonalities among use case required functions and properties) 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 such set 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 interworking of new management functionality and dynamic programmability of management functions. Evolution of the UMF core functions in terms of both number (e.g. new functions) and specifications is also possible and part of the modular and extensible design of the UMF. This evolution could for instance be driven by new needs or requirements that identify the necessity to add or re-design core function.

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). Such service orientation is already present in the current UMF design and specifications, although it is identified by the project that further investigation and thus a design/specification/implementation/assessment cycle is needed to better capture the service orientation implications, in particular with the recent initiatives started in different standards organization (e.g. ETSI, IETF) and fora on the network/service function chaining 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 either every well-formed goal injected to a network is 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 and enablers for the coordination and orchestration of the newly introduced self-x managing and managed entities. This can be based on both 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 (i.e. solve) 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 and knowledge indexing, 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 4**): 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.

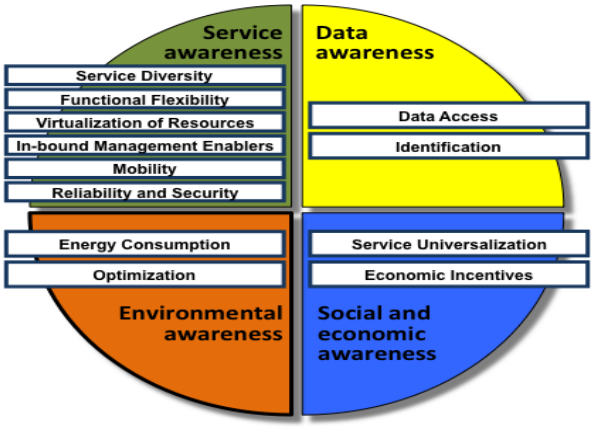


Figure 4. New Management Functionality for Future Networks

3. UMF functional specifications

3.1 UMF overview

Current and future networks are comprised of diverse autonomic network functions. UniverSelf targets autonomic management of network functions. The focuses of the project are to federate the network management over network segments and to bring autonomic interworking into its maturity age.

Unified Management Framework (UMF) is 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 interworking of autonomic functions (NEMs). **Network Empowerment Mechanism (NEM)** is 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)

The work in the project has followed two consecutive approaches: a top-down approach tackling autonomic management under a network wide approach from a somewhat theoretical point of view, and a bottom-up approach tackling many different autonomic solutions to network operators problem all across the different segments and layers of the network. This second approach is somewhat more pragmatic.

Considering together the two approaches, UniverSelf reached the conclusion that an autonomic management system for networks must afford the possibility of integrating autonomic functions coming from different vendors and integrating all them in a single management system. Regarding that, the UMF is the toggle point from a vision targeting autonomic management of network to its instantiation into the management of autonomic functions, which themselves manage the network.

Hence, the UMF is the framework to manage these autonomic functions in a safe way, which requires two things: first providing these autonomic functions with a given set of capabilities, and second to impose to these autonomic functions a given set of duties.

The previous release of the UMF design specifications [1], started with introducing the following picture of the UMF. It depicts, a management framework based on three main blocks namely, Governance, Coordination and Knowledge. The three together managing the autonomic functions named Network Empowerment Mechanism (NEM) once these autonomic functions comply with UMF specifications, i.e. once the software implementing the autonomic function is performing the duties imposed by the UMF specifications.

We should make sure that the choices made for the UMF are also discussed in section 1. The deliverable must be somehow self-contained since it the final release of UMF.

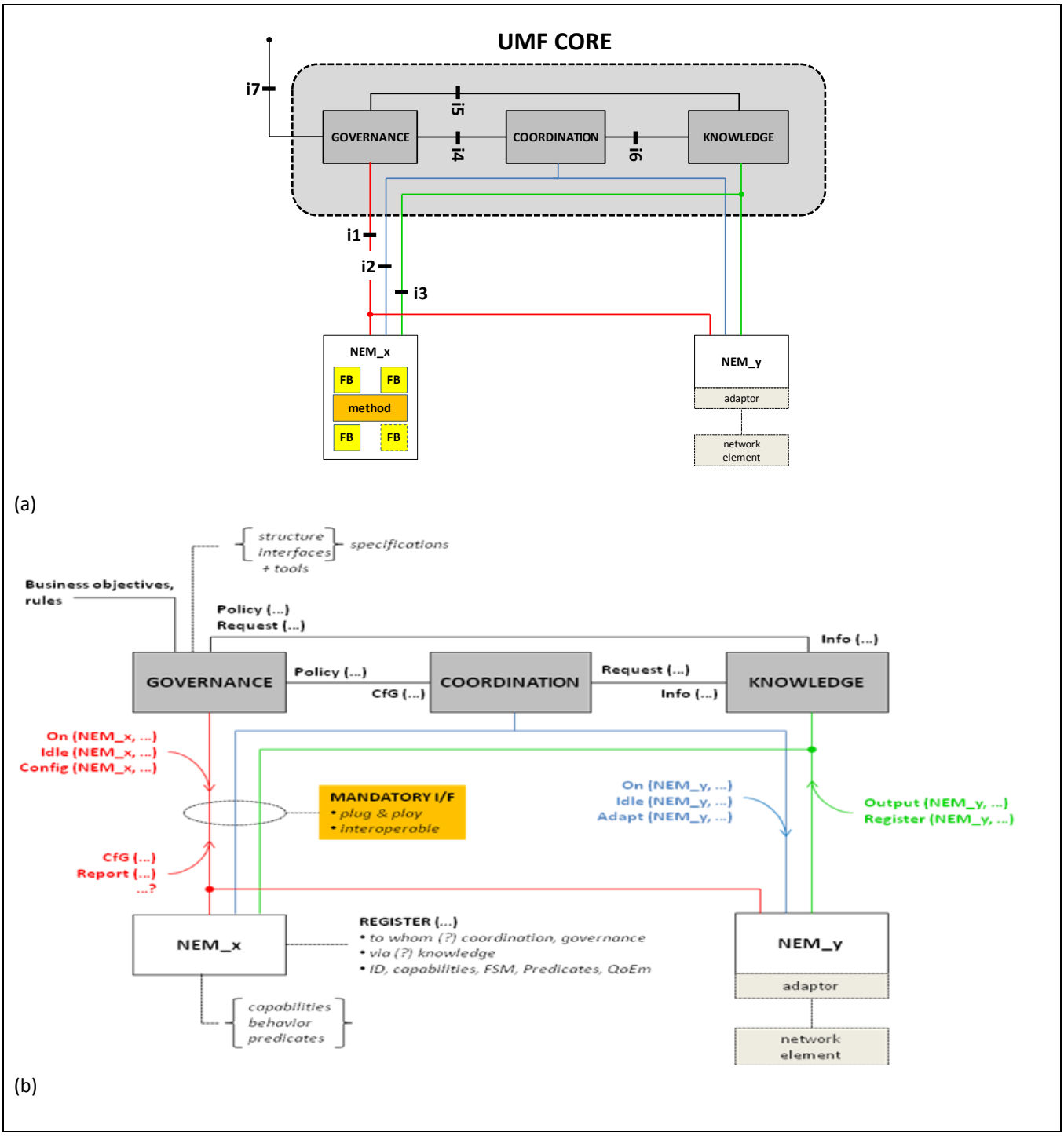


Figure 5. UMF overview and decomposition.

The introduction explained the paradigm of managing autonomic functions coming from a mix of providers, integrated in management system that can be implemented by some other providers. To achieve such a paradigm with pieces of software coming from different developers a standardized set of specifications detailing the capabilities and duties of each of the entities is mandatory. This is the rationale for the following section of this document, to pave the way for the standardization by defining the specifications of these entities, namely the UMF core blocks and the NEMs.

3.2 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):

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, any actor of the telecommunication/networking market can develop NEMs: 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 that 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).

A set of concepts related to NEM life cycle and its interaction with its outside world (UMF system) has been defined. It is therefore important to distinguish between:

- 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 performs a given autonomic function onto a given sub-set of network elements. 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 pieces of software running on different equipments. During runtime, the distinction between atomic and composite NEMs 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, as the deployment also requires the election of a leading entity.

Accordingly, the following machine-readable descriptions of the above concepts are explained below:

- 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 developer towards network operators,
- The **format 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. The NEM instance sends this description towards UMF system. This description is used for registration of the NEM. It tells which information is monitored and which range of actions can be taken.
- The **format 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 a set of instructions telling which network equipments or services MUST be handled by this NEM instance and which settings this NEM instance MUST work with. The GOV block of the UMF system is sending the NEM mandate
- 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 us sketch a very simplified process used to start an autonomic function (coming as a NEM class) inside a UMF system.

1. 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).
2. Second, the UMF GOV block is demanding to the installed software the creation of a NEM instance. Then the UMF GOV block sends a mandate for the NEM instance to deploy itself. The process ends with a NEM instance ready to register to UMF.
3. Third, this NEM instance is sending its NEM instance description to the all the three UMF blocks (GOV, COORD, KNOW) 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**.

The following 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.

3.2.1 Life-cycle of a NEM instance

A NEM from the moment that it is installed until the moment that it is uninstalled follows 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 6.

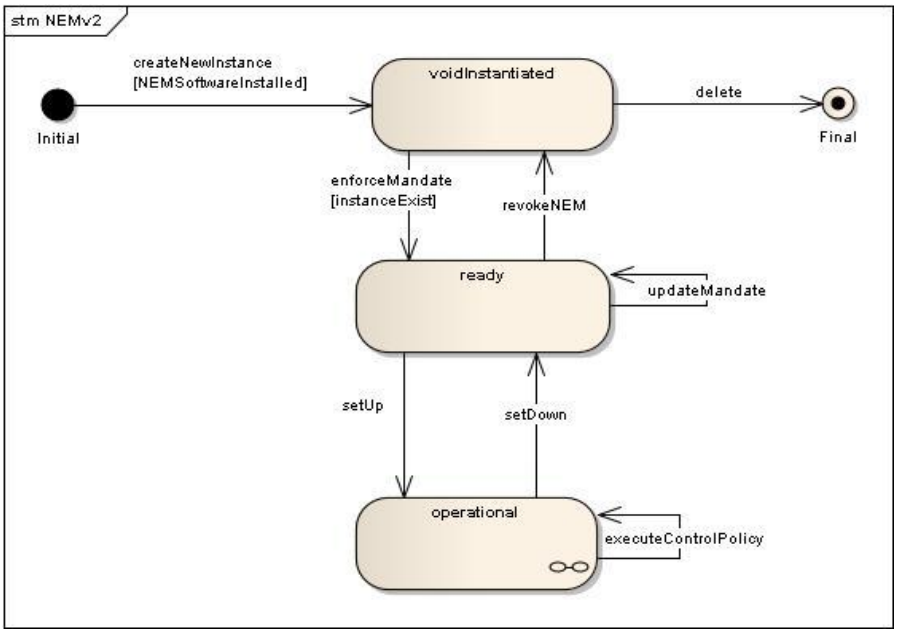


Figure 6. Simplified NEM instance life-cycle.

The NEM life-cycle consists of the following phases:

- INITIAL: 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 installed on relevant hosts.
- VOID INSTANTIATED: In this first state, the NEM exists as an instance. This state is mandatory, for a NEM instance to handle a MANDATE. The MANDATE is issued by the UMF system (GOV block) and determines the network resources that will be managed by this instance. The MANDATE also defines the configuration options[1] applicable to this NEM 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 allowed to set the mode of operation 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.

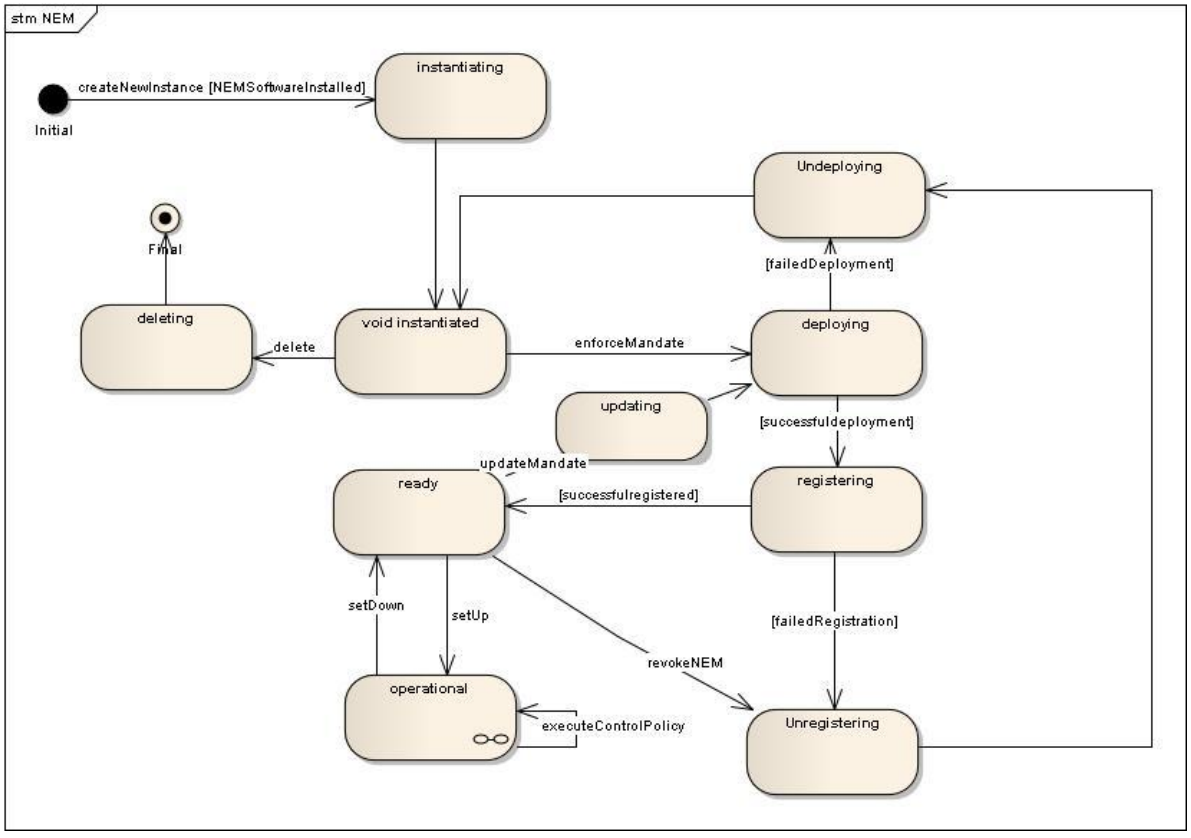


Figure 7. Detailed NEM instance life-cycle (with transitions).

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

When being created a NEM instance reaches the VOID INSTANTIATED state, where it 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 instance 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 stop all its processes, and then go back to the READY state, and notify COORD right after.

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 revokeNEM (from GOV), the NEM instance will reach the VOID INSTANCIATED 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 revokeNEM.

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 message.

This NEM life-cycle has been designed after state-of-the-art studies (e.g. OSGi [3] and SOAP) and analysis of MS26(Unification of the mechanisms embedding the UC methods [4]) material and extended to cover the specificities related to deployment of functions over distributed systems, knowing these functions can themselves be distributed. Sub-sections 3.2.3 - 3.2.6 describe the initial phase of the lifecycle (NEM Manifest), the NEM Installation and 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 in subsection 3.2.9.

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

3.2.2 Information model of NEMs

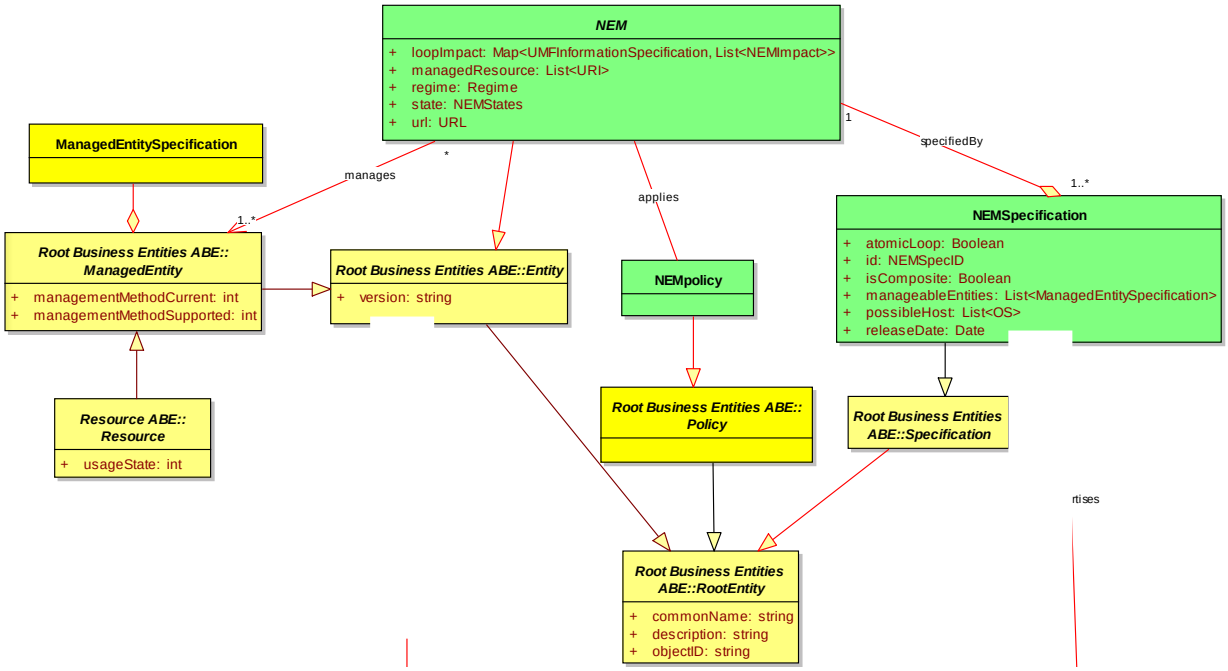


Figure 8. Inheritance of UMF information model from SID (NEM part).

Figure 8 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 [2].

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 NEMPolicy 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.

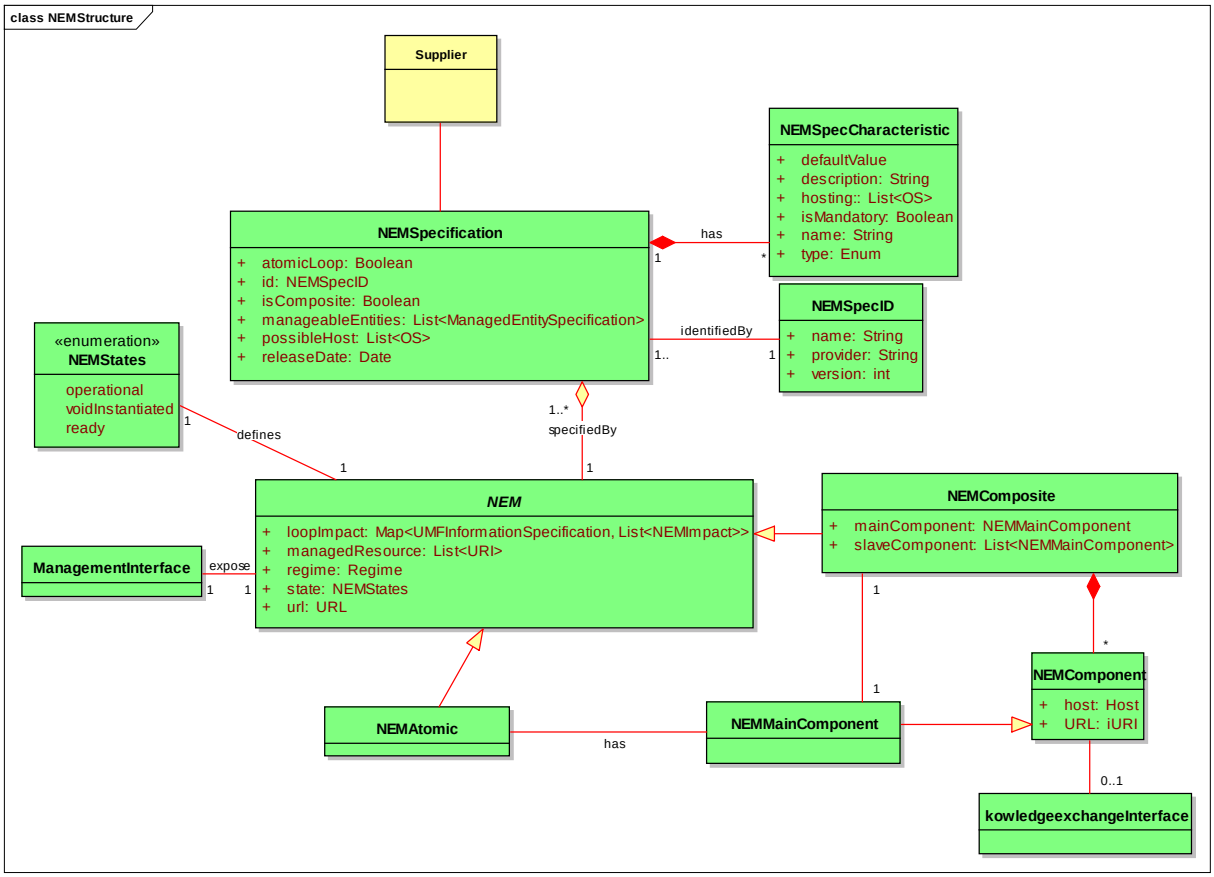


Figure 9. Representing the NEM structure in an information model view.

Figure 9 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

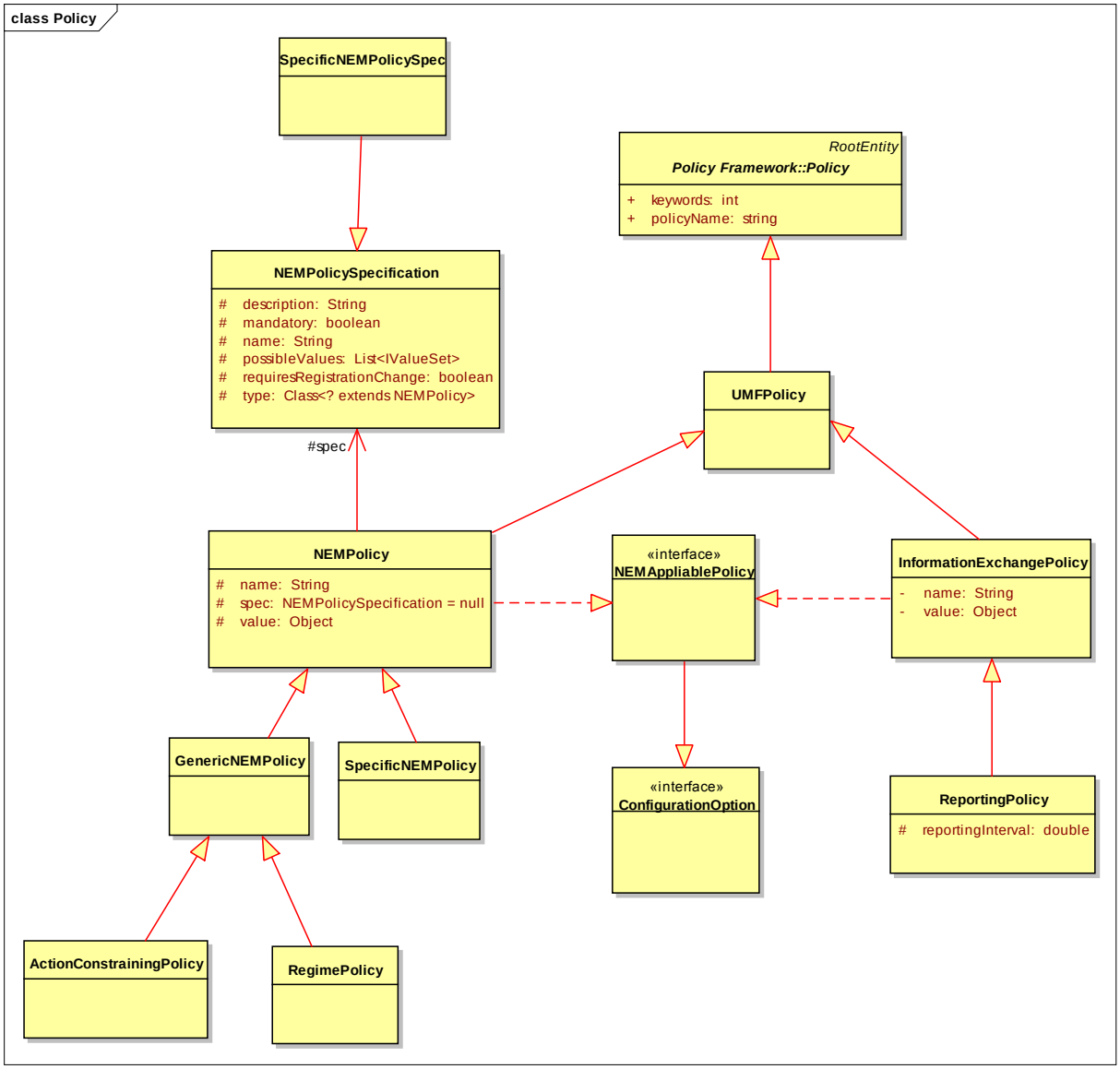


Figure 10. Information model of Policies regarding NEMs.

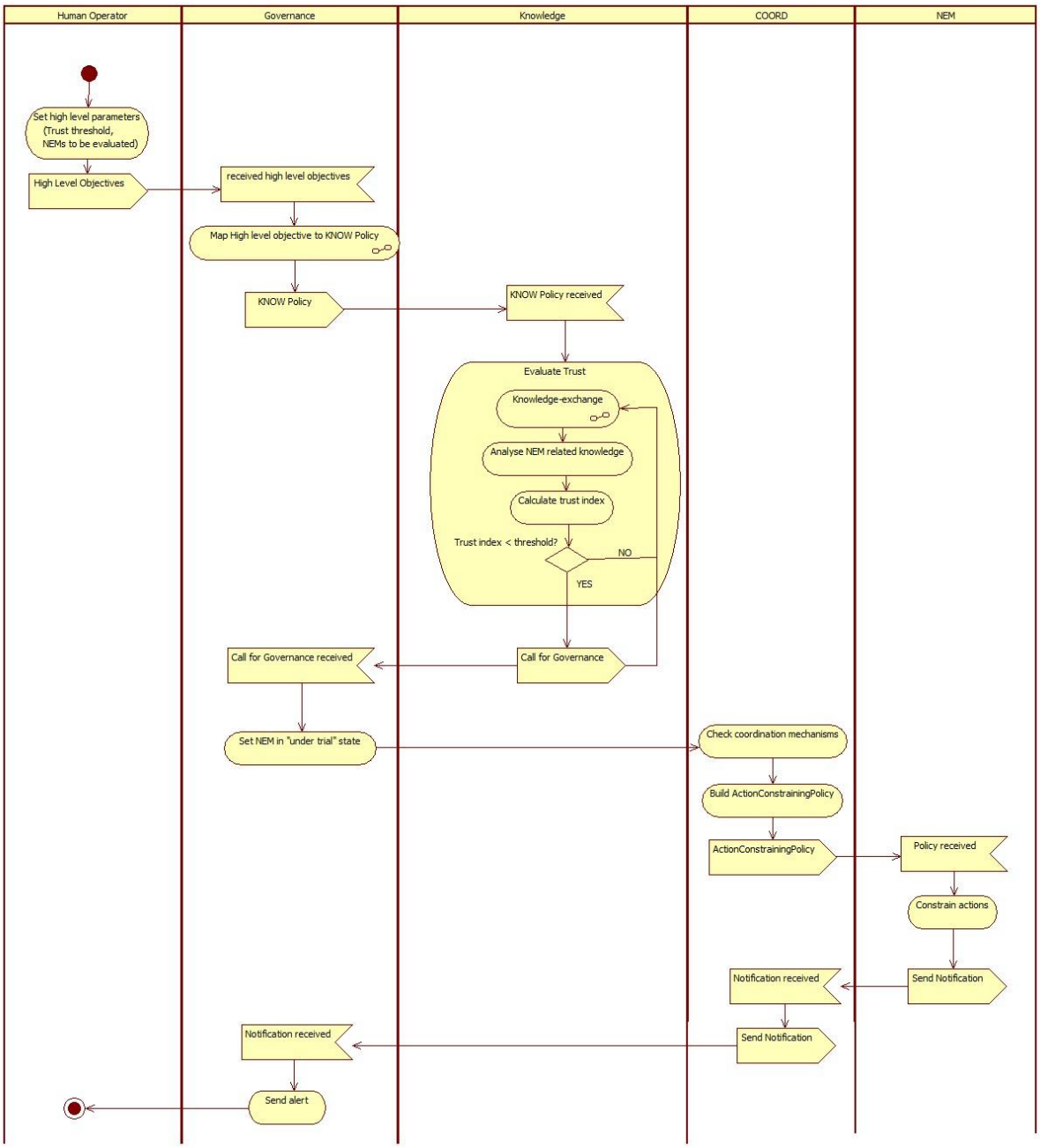
Figure 10 is depicting the inheritance of Policies in the scope of NEMs. First of all, all the policies are inheriting from UMFPolicy.

Then there are different types of policies:

- UMFPolicy is abstract and extends SID policy, it represents all the policies being specified by the UMF.
- GenericNEMPolicy is abstract, and represents all the kind of policies that are applicable only to any NEM instance, for which the format is defined by the UMF specification.
- 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. E.g., 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 0



- Figure 28: Call for Governance issued by Trust mechanisms

UniverSelf distinguishes between offline and online building-trust procedures. Online means that the operation of a given NEM or set of NEMs deployed at production level are being continuously scrutinized by the trust mechanisms during the operational phase of the NEMs. On the other hand, offline trust refers to an evaluation

made prior to the deployment at production level and therefore in conditions that may try to simulate or emulate the context in production. It is worth noting that in both cases the same trust-building mechanisms can be used, with the only difference of the actual context in which the NEM is embedded. The output of those mechanisms is a trust index, qualified as online or offline according to the scenario where the index was calculated.

- 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 specific format of its actual SpecificPolicies, are advertized in the NEM Manifest.

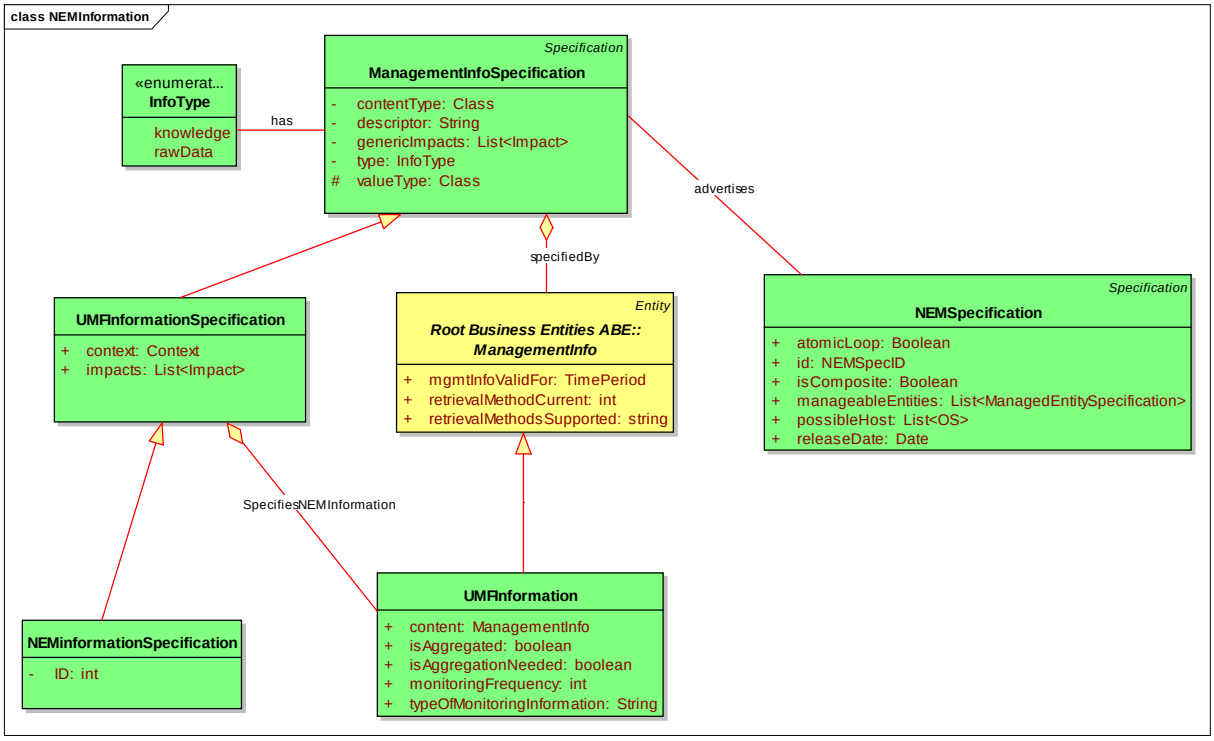


Figure 11. Information model of Information and Knowledge regarding NEMs.

Figure 11 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 3.3.5

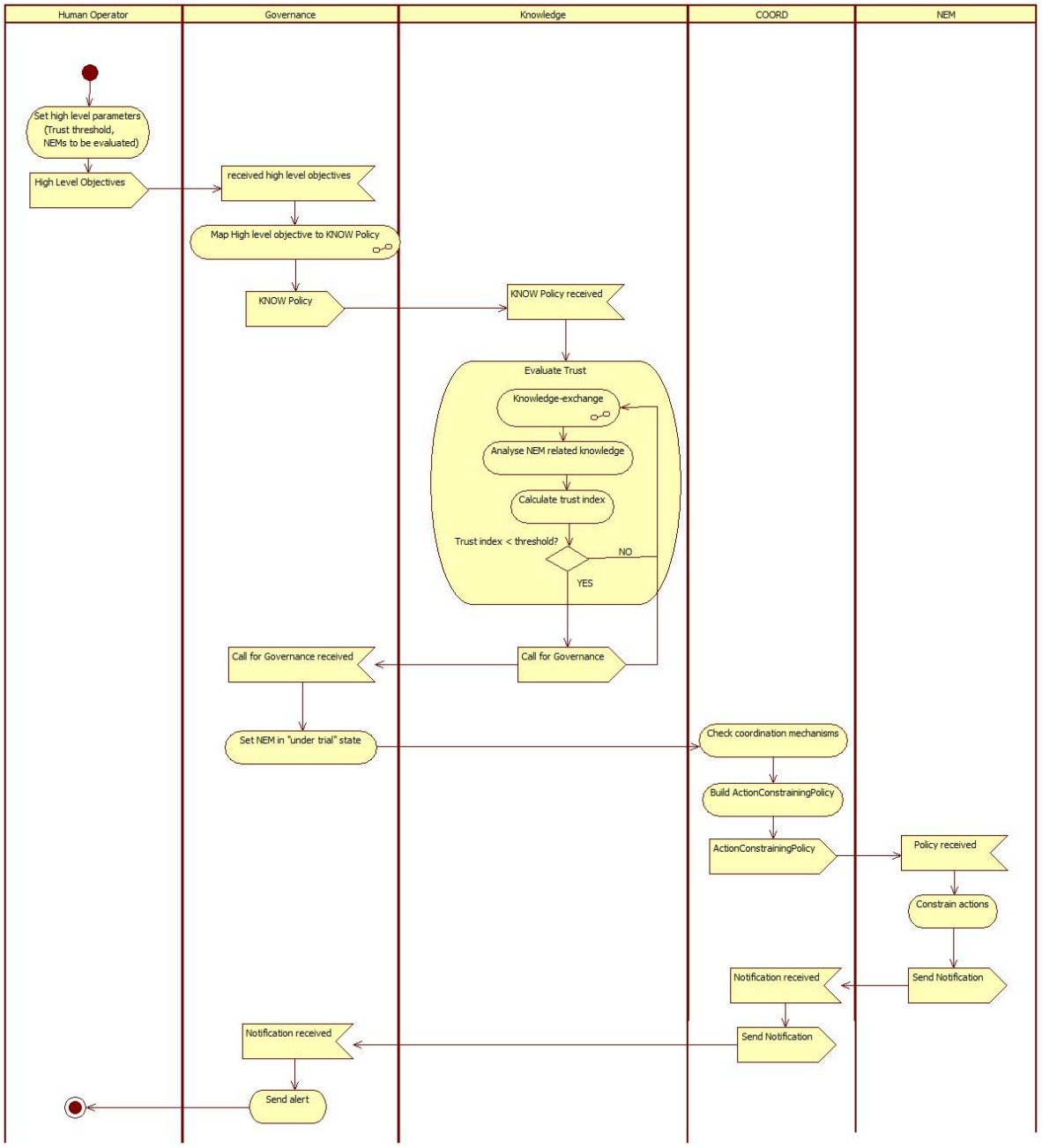


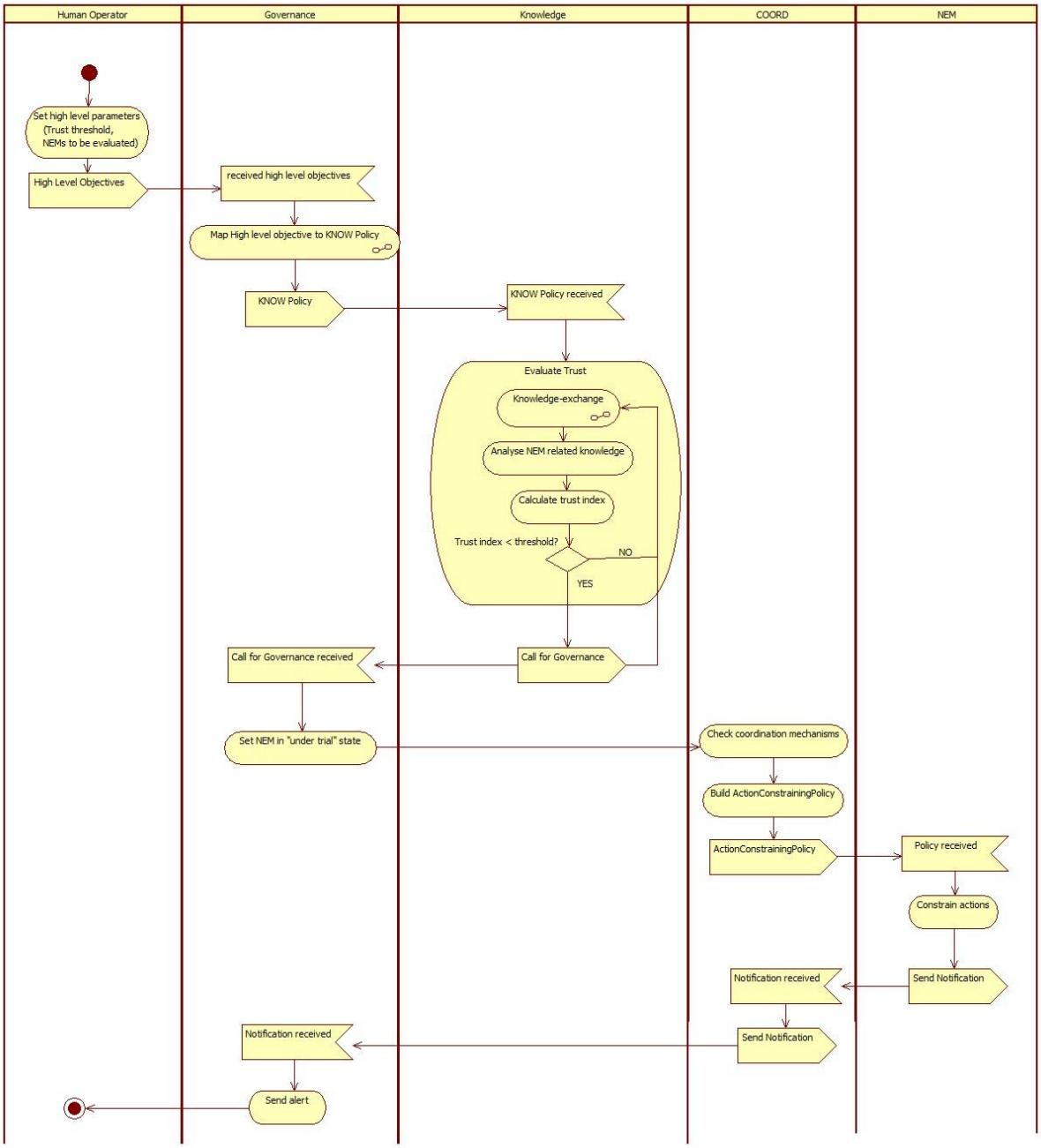
Figure 28: Call for Governance issued by Trust mechanisms

UniverSelf distinguishes between offline and online building-trust procedures. Online means that the operation of a given NEM or set of NEMs deployed at production level are being continuously scrutinized by the trust mechanisms during the operational phase of the NEMs. On the other hand, offline trust refers to an evaluation made prior to the deployment at production level and therefore in conditions that may try to simulate or emulate the context in production. It is worth noting that in both cases the same trust-building mechanisms can be used, with the only difference of the actual context in which the NEM is embedded. The output of those mechanisms is a trust index, qualified as online or offline according to the scenario where the index was calculated.

Information Flow Establishment and Optimisation function). A NEM can be at one or the two endpoints of such an exchange.

Figure 11 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 3.2.3), 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 0



- Figure 28: Call for Governance issued by Trust mechanisms

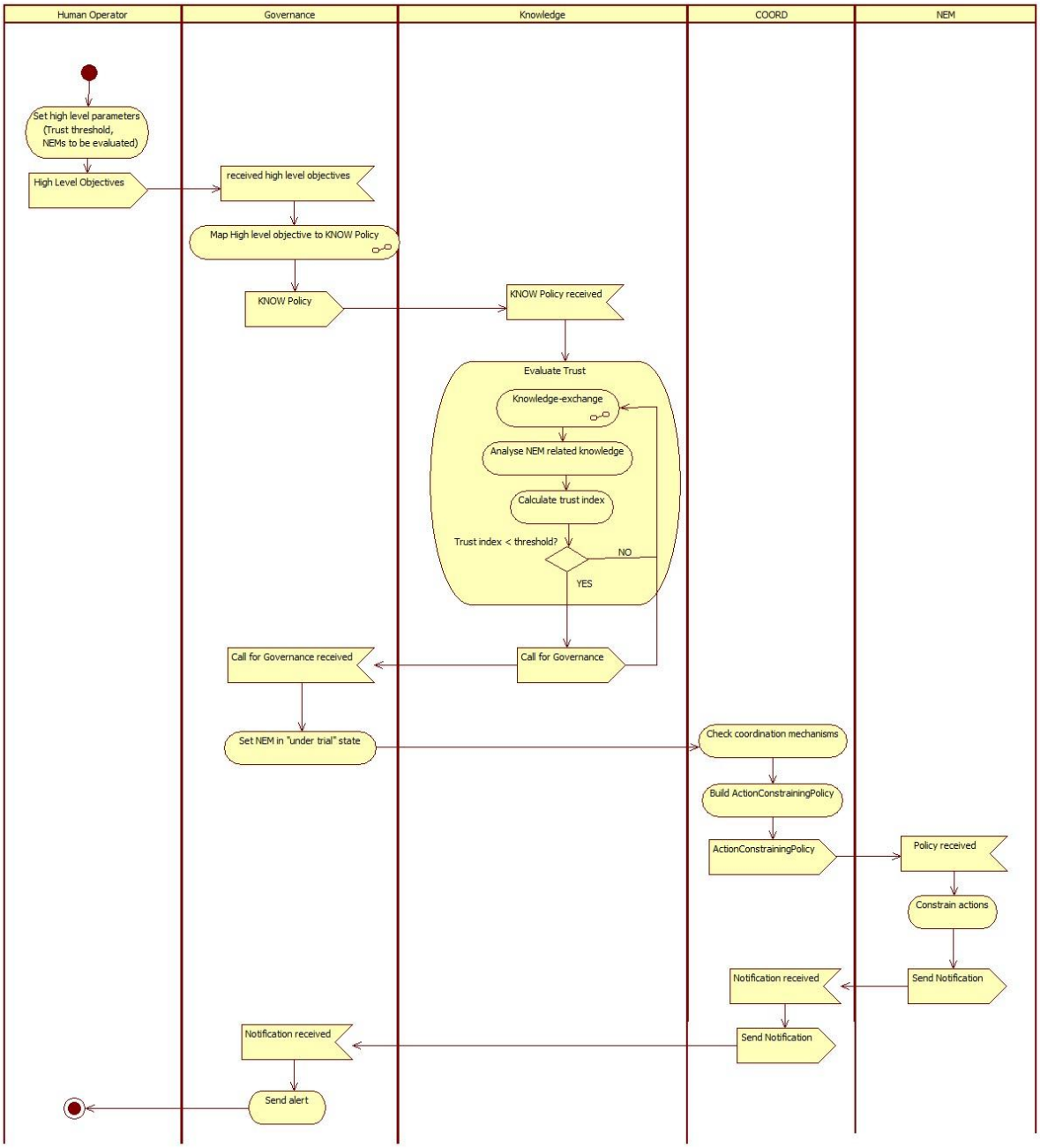
UniverSelf distinguishes between offline and online building-trust procedures. Online means that the operation of a given NEM or set of NEMs deployed at production level are being continuously scrutinized by the trust mechanisms during the operational phase of the NEMs. On the other hand, offline trust refers to an evaluation made prior to the deployment at production level and therefore in conditions that may try to simulate or emulate the context in production. It is worth noting that in both cases the same trust-building mechanisms can be used, with the only difference of the actual context in which the NEM is embedded. The output of those mechanisms is a trust index, qualified as online or offline according to the scenario where the index was calculated.

- 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 0).

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 [5].

3. UMFInformation: This class represents the information actually exchanged through a Knowledge Exchange Interface (see workflows in section 0



4.

5. Figure 28: Call for Governance issued by Trust mechanisms

UniverSelf distinguishes between offline and online building-trust procedures. Online means that the operation of a given NEM or set of NEMs deployed at production level are being continuously scrutinized by the trust mechanisms during the operational phase of the NEMs. On the other hand, offline trust refers to an evaluation made prior to the deployment at production level and therefore in conditions that may try to simulate or emulate the context in production. It is worth noting that in both cases the same trust-building mechanisms

can be used, with the only difference of the actual context in which the NEM is embedded. The output of those mechanisms is a trust index, qualified as online or offline according to the scenario where the index was calculated.

6. Information Flow Establishment and Optimisation function). For this exchange to happen KNOW takes in charge its organization, which will be materialized by an Information Exchange Policy (see Figure 10).
   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.

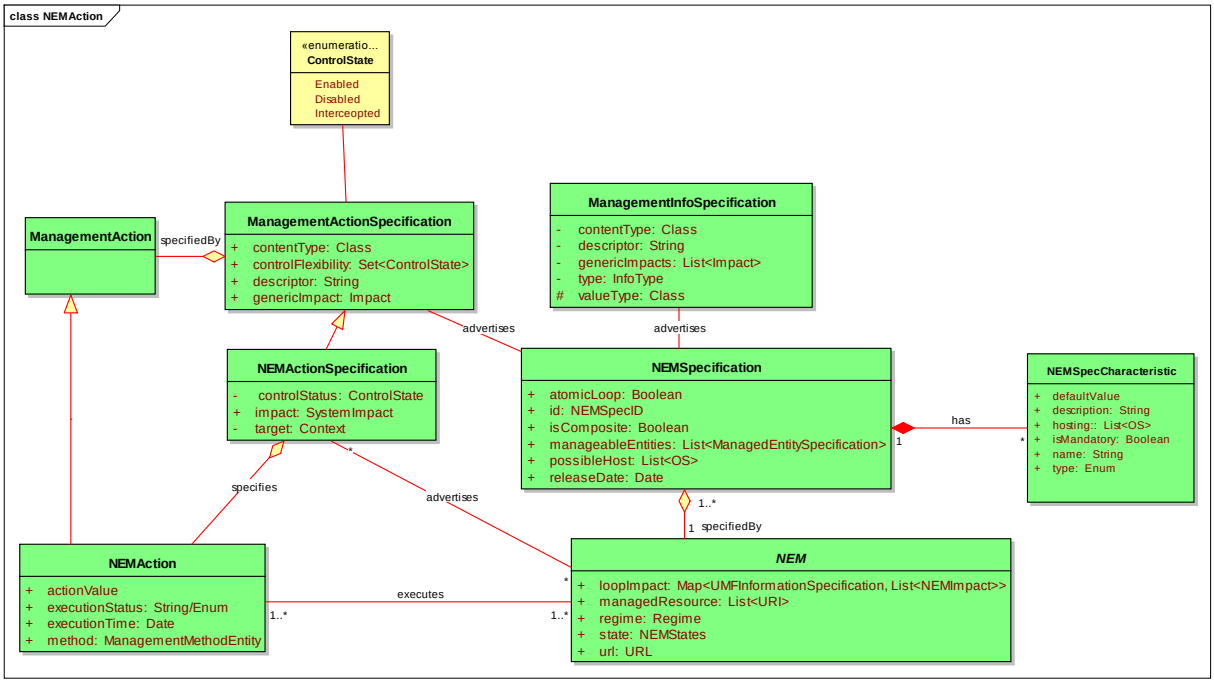


Figure 12. Information model of Actions regarding NEMs.

Figure 12 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 3.2.3).
   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 0).

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).

3.2.3 NEM Manifest

A NEM class is 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) that the operator will use 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 in order to modify at run time the initial settings of the NEM

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
Technology	List<TechnologySpecification>	List of the technologies to which the NEM is applicable
NetworkSegment	List<SegmentSpecification>	List of the network segments on which the NEM is applicable
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
FunctionalityFamily	List<Functionality>	List of the optimization targets of 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

Field Name	Type	Description
		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 Specific NEMPolicySpec - Name - Description - Event - Conditions - ConditionVariable - Operator - Default Value	List<Specific NEMPolicySpec> Specific NEMPolicySpec String String String List<ConditionAtomic> String int double	Lists the configuration options that can be applied to the NEM. The NEM policy specifications (e.g. critical parameters) must be depicted here.

More details on the classes of the fields of the Manifest are available in the information model section 3.2.2. Hereafter is an indicative example of the information which comprises a NEM Manifest, namely for the Green TE NEM, which 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.

```
<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>
   <Technology>
      <TechnologySpecification>IP</TechnologySpecification>
   </Technology>
   <NetworkSegment>
      <SegmentSpecification>Core</SegmentSpecification>
   </NetworkSegment>
   <PossibleHosts>
      <OS>Windows</OS>
```

```
    <OS>Linux</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 managed entity specification from the
    corresponding catalogue -->
</ManageableEntities>
<FunctionalityFamily>
    <Functionality>Load</Functionality>
</FunctionalityFamily>
<isAtomicLoop>true</isAtomicLoop>
<isComposite>false</isComposite>
<AcquiredInputs>
    <ManagementInfoSpecification>
        <descriptor>Description of router port(ID, capacity)</descriptor>
        <contentType>EthernetPortInfo<!--ID of a ManagementInfoSpec-->
        </contentType>
        <informationUsage>Acquired</informationUsage>
        <type>RawInfo</type>
    </ManagementInfoSpecification>
    <ManagementInfoSpecification>
        <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>
    </ManagementInfoSpecification>
    <ManagementInfoSpecification>
        <descriptor>Load of router interface</descriptor>
        <contentType>Numeric</contentType>
        <informationUsage>Acquired</informationUsage>
        <type>RawInfo</type>
    </ManagementInfoSpecification>
    <ManagementInfoSpecification>
        <descriptor>Routing Table</descriptor>
        <contentType>List<LSA></contentType>
        <informationUsage>Acquired</informationUsage>
        <type>RawInfo</type>
    </ManagementInfoSpecification>
</AcquiredInputs>
<OptionalExternalInputs>
    <ManagementInfoSpecification>
        <descriptor>Prediction of router interface load</descriptor>
        <contentType>Numeric</contentType>
        <informationUsage>External Optional</informationUsage>
        <type>Knowledge</type>
    </ManagementInfoSpecification>
    <ManagementInfoSpecification>
        <descriptor>Prediction of router interface load</descriptor>
        <contentType>Numeric</contentType>
        <informationUsage>External Optional</informationUsage>
        <type>Knowledge</type>
    </ManagementInfoSpecification>
</OptionalExternalInputs>
<PossibleActions>
    <ManagementActionSpecification>
        <descriptor>Switch ON/OFF Ethernet port</descriptor>
        <contentType>Boolean</contentType>
        <controlFlexibility>{Enabled, Disabled, Intercepted}</controlFlexibility>
    </ManagementActionSpecification>
    <ManagementActionSpecification>
        <descriptor>Switch ON/OFF Ethernet port</descriptor>
        <contentType>Boolean</contentType>
        <controlFlexibility>{Enabled, Disabled, Intercepted}</controlFlexibility>
    </ManagementActionSpecification>
    <ManagementActionSpecification>
        <descriptor>Switch ON/OFF IP interface</descriptor>
```

```
            <contentType>Boolean</contentType>
            <controlFlexibility>{Enabled, Disabled}</controlFlexibility>
        </ManagementActionSpecification>
        <ManagementActionSpecification>
            <descriptor>Change metric of IP interface</descriptor>
            <contentType>Numeric</contentType>
            <controlFlexibility>{Enabled, Disabled, Constrained}</controlFlexibility>
        </ManagementActionSpecification>
    </PossibleActions>
    <ConfigurationOptions>
        <SpecificNEMPolicySpec>
            <Name>Condition_00001_MPLS_TE</Name>
            <Description>Load Threshold reached to trigger MPLS TE</Description>
            <Event>OnGovPolicyUpdate</Event>
            <Conditions>
                <ConditionAtomic>
                    <ConditionVariable>CurrentLoad</ConditionVariable>
                    <Operator opType="2"/>
                    <DefaultValue>0.2</DefaultValue>
                </ConditionAtomic>
            </Conditions>
        </SpecificNEMPolicySpec>
    </ConfigurationOptions>
</eu.univerself.nem.Manifest>
```

3.2.4 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[4] of a given type of network equipments/resources,
2. The code of the NEM is manually[5] 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 current UMF release 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 CreateNewInstancemessage 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[6] 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] The software controlling the hardware.

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

[6] The reason is to allow these loaders to pick the best for the instantiation of the NEM.

- 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 CreateNewInstancemessage, 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 CreateNewInstance 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

3.2.5 NEM Mandate

A **NEM mandate** is issued by the UMF system (GOV block) to a NEM instance. This NEM Mandate is a set of instructions telling which network elements and resources and services MUST be handled by this NEM instance and which settings this NEM instance MUST work with. Moreover, NEM mandate provides to the NEM instance the needed information for exchanges with all UMF blocks (GOV, KNOW, COORD).

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 elements/resources/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>
<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>
</Configuration Options>
</eu.univerself.nem.Mandate>
```

The deployment of a NEM instance MUST happen accordingly to the NEM Mandate. When receiving the NEM 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 3.2.1), a NEM Mandate can be sent to a NEM instance, when:

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

The NEM Mandate determines a list of network elements, resources and services. 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 network elements,
- additionally, federating these software components into a unique entity by the selection of the entity acting as the unique communication point with the UMF system.

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 NEM instance description.

3.2.6 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 all the three blocks of UMF system. This description is used for registration of the NEM. It tells which information are monitored and which 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

More details on the classes of the fields of the Instance description are available in the information model section 3.2.2. 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 3.2.5.

```
<eu.univerself.nem.InstanceDescription>
    <NEMspecID>
        <Name>Green TE</Name>
        <Provider>StylianosCorp</Provider>
        <Version>1.0.0</Version>
    </NEMspecID>
    <Instance ID>356789456</Instance ID>
    <Management@>100.200.1.15</Management@>
    <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>
```

3.2.7 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.

3.2.8 NEM's Mode of operation

COORD block is controlling the behaviour of NEM instances when those are in the OPERATIONAL state. The main way of controlling the NEM consists in enforcing the mode of operation of the NEM instance. This is achieved by setting a RegimePolicy to the NEM that specifies a regime.

The information model of the NEM regime describes how the mode of operation of the NEM can be enforced, mainly according to three set of parameters:

1. A priority parameter, to give an indication on the priority of processes between NEM instances competing over the same host resources.
2. A completion parameter determining how much of the MAPE loop is performed (Monitoring, Analyzing, Planning Executing). E.g. a network operator may want to test a NEM without having it executing its action, what is sometimes named an under-trial mode. This happens by GOV sending a Policy to COORD specifying which NEM instances must be in such a mode, then COORD would only apply Regimes to these NEM instances with a completion level that is not containing the execute phase.
3. A set of parameters related to time of execution of the MAPE loop, this allows specifying a time window in which to run a period between two runs etc…

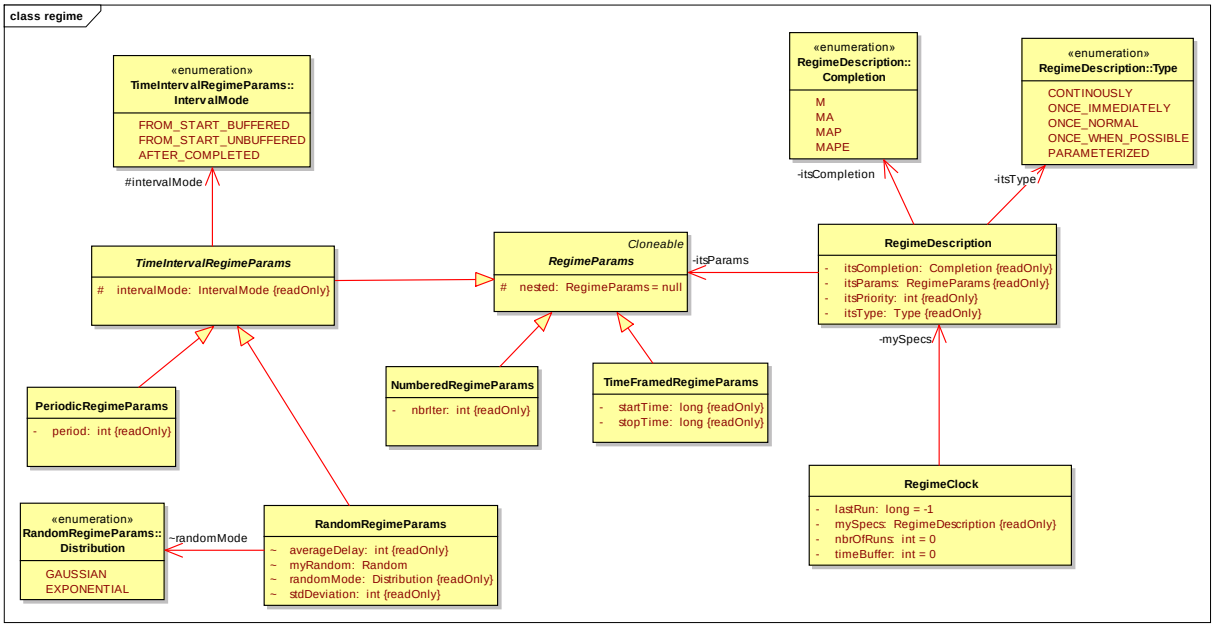


Figure 13. Information model of NEM regimes

3.2.9 Description of the operations for state transitions

The operations that a NEM is exposing to the UMF are the following.

Operation Name	getState
Description	Retrieves the current state of the NEM instance.
Constraints	The NEM instance 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: INITIAL, INSTANTIATING, VOID INSTANTIATED, DEPLOYING, UPDATING, REGISTERING, READY, OPERATIONAL, UNREGISTERING, UNDEPLOYING, DELETING
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 3.2.3)
List of non-functional requirements	

Operation Name	enforceMandate
Description	Sets the NEM mandate for the NEM (see Section 3.2.5). 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 3.2.5)
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. Additional Info 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 3.2.5)
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.

3.3 Governance block

Governance block responds to the need of the human network operators to have the possibility of supervising the functioning and controlling the behaviour not only of the underlying autonomic functionalities (NEMs), but of the management system as well (UMF core blocks).

Four main functions were identified as necessary for the proper governance of future networks and services:

- Human to Network (H2N) function
- Policy Derivation and Management (PDM) function
- NEM Management function
- Enforcement function.

The Human to Network function provides a set of operations that allow the human network operator to manage the whole autonomic network, services and the other components of the UMF system by editing and validating high level objectives. A high-level objective can be defined as an overall target that the network must achieve. These objectives are expressed in a close-to-natural language and therefore need to be translated to a language that can be understood by the targeted elements. This translation process is implemented in the Policy Derivation and Management function, which produces as output low-level policies. Enforcement function transmits then the final policies to the corresponding NEM or UMF component, while NEM Management function allows a fine-grained control of the NEM life cycle.

This chapter elaborates the specification of the above-mentioned functions and related operations that conform the Governance core block. This chapter builds on the Governance specification in Deliverable D2.2, and presents a more consolidated version, especially with respect to the relationship with the other UMF blocks and functions, and the description of input and output parameters for each of the operations.In particular, a mapping of the operation's parameters to the SID-based UMF information model is provided.

3.3.1 Human to Network function

The Human to Network function provides means for injecting input to the network through the definition of high level objectives, and at the same time, for receiving information about the functioning of the network, services and the UMF system. In order to reach these goals, the Human to Network Interface function has been decomposed into a set of operations: Service Definition, High Level Objectives Definition and Supervision. The first two operations offer the programming functionality of network, services and the UMF system, while the later provides feedback about their current status. A detailed description of each of the above mentioned operations is provided in the following subsections.

Service Definition

A service instantiates a specific functionality offered to a user by the execution of one or more applications. Service Definition operation is about the creation and representation of service descriptions. This description will be translated later on into terms that allow a NEM to understand the requirements of individual services, such as QoS (Quality of Service), availability or security. The definition of a service describes its invariant characteristics, which requires the specification of a set of parameters needed for the delivery and proper operation of the service:

- Service name and Service Identification Number
- Service description: textual description of the service.
- Resources: description of the equipment types needed to run the service, and its requirements of hardware, protocol and software, as well as management information for those devices.
- NEM classes: identification of the NEM classes that will be instantiated when the service is deployed.
- Location: specification of the geographical area where the service will be available.
- Service characteristics: list of parameters that describe the service, i.e. redundancy level or availability.
- Service characteristics values: set of attributes that describe the values of the parameters that a corresponding Service characteristic object can take on.
- Service performance parameters: KPIs that will be used to measure the performance when the service is running. These are the parameters that NEMs will have to monitor and communicate to KNOW.

- Service templates: A service template represents a type of Service Specification introduced for the purposes of fulfilment. It defines specific service characteristic values that can be dynamically referenced by multiple service instances during their lifecycle span. Several templates can be produced for the same service, corresponding to different levels (e.g. Gold, Silver and Bronze levels).
- Service level objectives: A set of quality goals for the service defined in terms of parameters and metrics, thresholds, and tolerances associated with the parameters. Two types of parameters must be considered here: KPIs (Key Performance Indicator) and KQIs (Key Quality Indicators). KPI measures a specific aspect of the performance of a resource or a group of resources of the same type. A KQI measures a specific aspect of the performance of a service, and draws its data from a set of resources.

These concepts are modelled in the Service related ABEs (Aggregate Business Entities) of SID model, and therefore they are inherited by the UMF information model. Additionally, the deployment and operation of the defined service need the identification and suitable configuration of the NEMs able to manage the network resources required to deliver the service. This identification process can seamlessly take place following the information contained in the NEM Manifest, which includes a field for the list of the entities a NEM can manage. This mash-up is achieved by the NEM Management function, specified in Section 3.2.3.

Table 6. Service Definition operation

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 : Service name, Service description, Resources, NEM classes, location, Service characteristics, Service characteristics values, Service performance parameters, Service templates, Service level objectives (see description above)
List of output data	Service
List of non-functional requirements	

High Level Objectives Definition

One of the operations provided by the Human to Network function is the definition of technology agnostic management objectives using a high-level language. These objectives can be used to fine-tune the network behaviour and the configuration of the UMF core blocks. These high-level objectives correspond to:

- Service characteristic values. Service demands concern requests relevant to service provision, such as the delivery of a new service to a given customer with given service characteristics, the accommodation of new traffic in a particular specific geographical area and time period, or the modification of already active service quality conditions. Service configuration assigns values to already predefined service characteristics (e.g. redundancy level or availability).
- Configuration of UMF core blocks which allow the customization of the behaviour of KNOW and COORD blocks.

These high-level objectives need to be further propagated to the targeted elements (NEMs or UMF core components), which requires their prior transformation into lower level policies. This translation process is accomplished in the Policy Derivation & Management function, described in Section 3.3.2. After the translation process, low level policies will be generated that correspond to:

- NEM configuration options: These are policies sent to NEM, in order to set the configuration of the NEM, and then logically its behaviour. Hence, the modification of the network behaviour is achieved through the NEMs, which act on their managed resources. The different configuration options are

specified in the NEM Manifest of each NEM class. Examples are the activation of an energy efficiency level or the alteration of load balancing requirements in the network.

- Coordination configuration options, which allow the modification of the COORD behaviour. These parameters are strongly related to the policies needed by the Coordination core block. Specific COORD Policies can specify three types of configuration options :
    - Conflict type, that defines what COORD will consider as a conflict to be addressed:
        - Parameter: a parameter conflict refers to the situation where two or more NEMs would need to control the same network parameter
        - Metric: a metric conflict refers to the situation where another NEM B. uses the output of a NEM A. This can be problematic when NEM B is executing a learning algorithm and NEM A invalidates the assumptions that NEM B is making about its inputs (e.g. uncorrelated traffic, stationary traffic).
        - Loop: where a cascade of NEM outputs leads back to the originator of the cascade (NEM A). This can be problematic if the cascade of outputs evolves in a way that leads to frequent changes to the input of NEM A that will require an action from NEM A in the opposite direction of its previous action (oscillations)
        - All: commands COORD to consider the three types of conflicts above.
    - Sensitivity, as a value [0, 1]; This configuration option defines how sensible COORD can be when identifying potential conflict situations, how to assign mechanisms and also on how to interpret monitored information. Some examples are listed below :
        - when identifying parameter conflict it can determine if contradicting parameter changes may be tolerated and the maximum allowed difference.
        - when identifying a loop it can determine the length of the "path" that will be searched to check if it leads back to the same NEM
        - when identifying a metric conflict it can determine when the assumptions of a NEM are invalidated by the output of another NEM
    - Orientation, to be used by COORD to prioritize/weight NEMs according to operator performance objectives. It can be set on a per-segment basis and requires that NEMs also include their orientation in their manifests. For instance, if a NEM A states as orientation “Energy” and another NEM B as orientation “Load Balancing”, and GOV in general states that “Energy”>>”Load Balancing”, then the priority assigned by COORD to NEM B will be lower than the one assigned to NEM A.
- Knowledge configuration options, which allow the modification of the KNOW behaviour. Examples of these configuration options are the definition of optimization goals to Knowledge block to reduce the communication overhead, or the definition of a threshold for the detection of untrustworthy NEMs. These parameters are strongly related to the policies needed by the Knowledge core blocks:
    - OptimizationGoalsPolicies define optimization objectives that cause KNOW functions to adapt their tactics in order to meet the goals. Examples of these optimization goals are :
        - Minimise communication overhead.
        - Maximise energy efficiency.
        - Minimise response time.
        - Maximise robustness.
    - TrustPolicies are used to configure the trust mechanisms that continuously assess the NEMs behaviour (see Section 3.3.4.1 for details on trust mechanisms). The information in the policies includes the list of NEMs to be evaluated, and the conditions for the evaluation (e.g. a given time period or a trust index threshold) :
        - Evaluate trust on all NEMs by default (global policy)
        - Evaluate this NEM or this group of NEMs until they reach a trust index of 0.9
        - Evaluate this NEM or this set of NEMs for 10 months

- Set trust index threshold for this NEM or this set of NEMs to 0.85. When the trust index drops under the specified threshold, the trust mechanisms will inform GOV through the appropriate interface.

Table 7. High Level Parameters definition operation

Name	High Level Objectives Definition
Description	High Level Objectives Definition block allows the definition of objectives for a given NEM or group of NEMs, service or group of services, or even the UMF components.
Constraints	
List of input data	Service characteristics, NEM Manifest
List of output data	Service characteristic values, NEM Configuration values, Coordination/Knowledge Configuration values
List of non-functional requirements	

Supervision

Previous sections described the operations of the Human to Network function that facilitate the communication of the human network operators towards the UMF core, network elements and NEMs. Conversely, the H2N function should provide information about the status of the above mentioned elements. This information refers to the configuration, status, context and alerts of networks, services, NEMs and UMF core. The Supervision operation provides the human operator with a clear picture of current network and system conditions. Based on that, he may decide to use the other operations to modify the behaviour of the network or system.

The importance of the decision about which information must be shown in order to fulfil the requirements from the human network operators should not be underestimated. The rest of this section does not aim to specify how this information should be displayed in a graphical user interface, but to determine the minimal amount of information that any UMF-based Governance implementation should provide, namely:

- Alerts. Even when NEMs are designed to be autonomous, in rare occasions they can reach a situation where they are not able to fulfil the specified goals in their current context. In those cases, where all the self-healing attempts have failed, the NEM should scale up the problem. In some circumstances, the misbehaviour of a set of coordinated NEMs may be the origin of the problem, and therefore COORD is informed in the first place. If all the feasibility checks of the coordination mechanisms fail, then the human network operator should be alerted.

  The information attached to each alert should contain all the details needed for the human operator to take control over the situation:

  - Importance factor: an indication of the impact of this alert on the functioning of the network. Alerts should be prioritized, so those of highest severity level should be informed first.
  - Priority: An indication of the importance of the resolution of the alert.
  - Originating system: an indication of where the problem was generated
  - Description: textual description of the alert
  - Category: classification of the alert
  - Responsible: person, organization or entity in charge of resolving the problem
  - Start time: time at which the alert was raised for the first time
  - Change time: time at which the alert was last changed
  - Related alerts: link to related alerts. Related alerts may be parent alerts (a parent problem from which the alert derives) or underlying alerts (alerts derived from this one)
  - Status: indicates if the alert is active, closed or under investigation
  - Root cause of the alert: to be filled when the problem that caused the alert is diagnosed
  - Affected resources and services: list of resources or services that are affected by this alert

The alerts directly related to services and resources correspond to the ServiceProblem and ResourceAlarm classes in SID model, while the alerts related to NEM problems are defined as an extension to SID model (see Figure 14). A NEM problem may affect the services, the resources and the UMF core mechanisms, and can be specialized into three types of problems:

- UMFToNEMInteractionProblem describes the interaction problems, Time-out and Service unavailable. It could be the NEM sending wrong messages instead of precise ones, or NEM not responding to UMF requests/messages.
- NEMAlgorithmProblem describes a faulty behavior of the NEM that do not corresponds to its mandate, do not respect UMF policies and do not respect what was declared within the manifest.
- NEMToNetworkProblem class describes the possible problems occurring between the NEM and the underlying resources it is managing. Two possible specializations for this class: The NEMToNEtworkCommunication describes the interaction fault, the time-out and NEM unavailable; the NEMSemanticProblem describes the real time violation, the QoS problems and the SLA related problems.

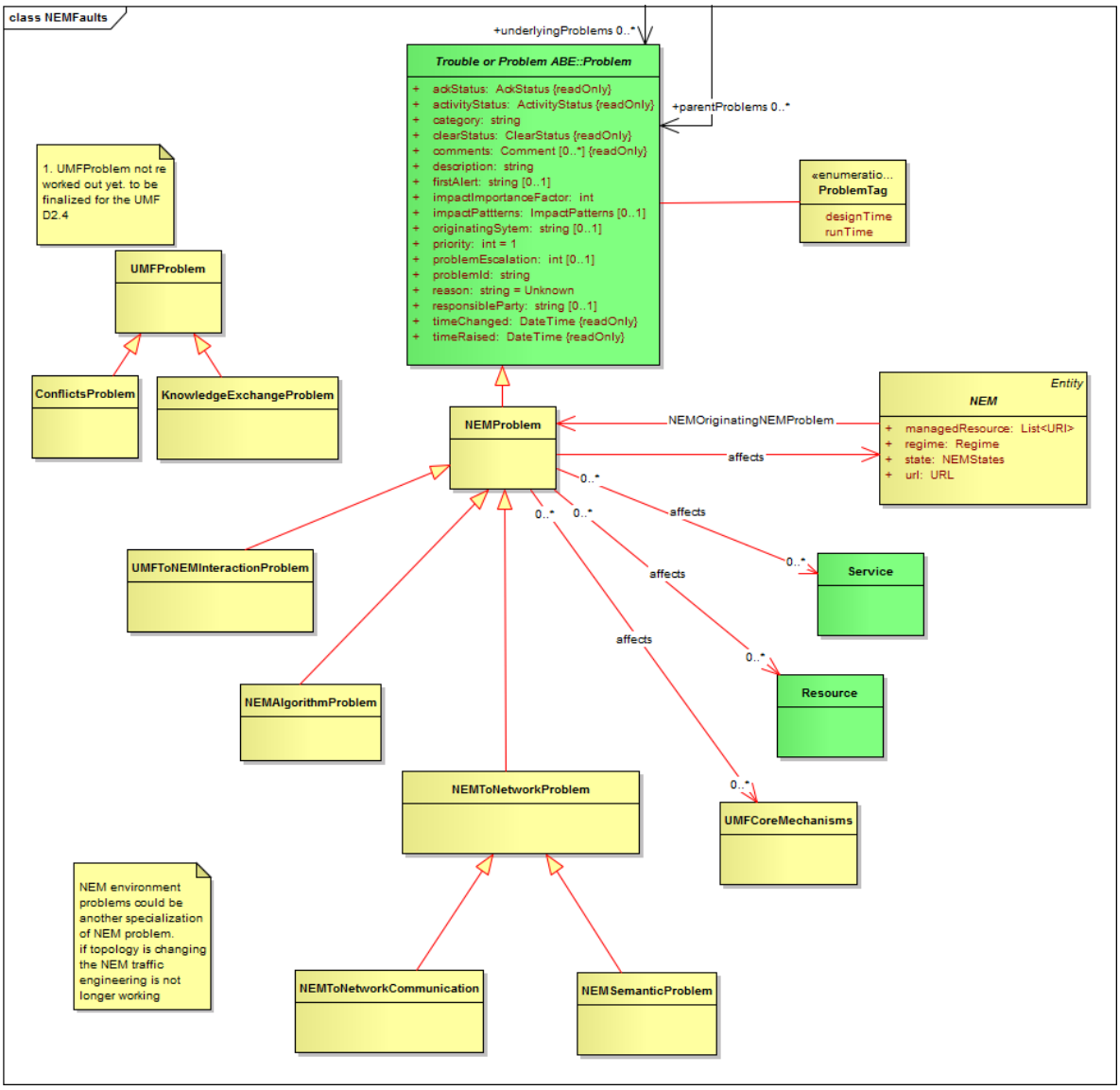


Figure 14: NEM Problem classes

Alerts described above include alarms from network elements, notifications triggered by NEMs when a network failure is detected and information about malfunctioning of UMF core system. The gathered data from all these different sources guarantees that, in case of a failure that cannot be automatically recovered, the human network operator has access to all the relevant information produced by the network elements, NEMs and the UMF components. By default, a summary of the active root alerts should be organized in increasing order of priority, being the most impactful ones the first to be noticed by the human operator. Root alerts are those that do not have parents, and therefore may be causing other alerts. The human operator should also have the possibility of requesting detail information for each of the alerts, including the underlying ones, and to query past alerts and alerts with other statutes than the active one.

- Performance parameters. Performance measures the manner in which a service, network resource or software component is performing or has performed. An overall view of the system status needs a monitoring infrastructure that expands to network elements, services and the UMF core. For each of them, the following information must be provided:
    - Identification of the monitored resource, service, NEM or component
    - Description: definition of the performance attributes
    - Time when the monitoring data were collected
    - Characteristic Specifications: definition of the monitoring parameters. A characteristic can take on a discrete value, can take on a range of values, or can be derived from a formula which description is included in the attributes of the entity.
    - Measured values: the actual values that the characteristic specifications take
    - Performance Indicator Specification: a measure of a specific aspect of the performance of an entity, including the derivation algorithm or formula used to calculate it. These indicators include derived parameters but also diagnosis or predicted values as well.
    - Performance values: the actual values that the performance indicator takes.

  This information corresponds to the Performance ABE in SID.
- Action taken: A NEM should be able to detect and recover from potential problems and continue to function smoothly. Based on the collected monitoring information, a NEM may diagnose or predict failures in the network, and then implement remediation actions (corrective or preventive). These actions must be stored in KNOW, together with all the information that led to the conclusion that triggered the action. Human network operator should be able to retrieve this information on demand.
    - Description: textual description of the action
    - Category: type of action (corrective or preventive)
    - Target: network resource(s) on which the action will be implemented
    - Affected elements: network resources and services that will be impacted by the action
    - Execution time: time at which the action was triggered
    - Planned execution time: the time when the planned action should take place
    - NEM id: identification of the NEM who proposed the triggering of the action
    - Current status of the action: an action may be in planned state (not yet executed), in execution, or terminated. In case of NEMs in “under trial” state, the actions will never be implemented, and therefore there will be in planned state forever.
    - Context: all the relevant monitoring parameters that led to the triggering of the action

  This information corresponds to the NEMAction class in the UMF information model.
- NEM classes information. The human operator needs to access the inventory of available NEM classes:
    - Characteristics as defined in the Manifest: features, possible hosts, possible actions, needed inputs and provided outputs, and configuration options
    - Certification level. Typically, any network element or network management system must pass certification tests before being deployed at production level. A NEM certificate summarizes the result of such evaluation, after having assessed the behaviour of the NEM, its performance, and the compliance with UMF.

- Aggregated information about the offline and online trust indexes of the NEM instances. Online trust means that the operation of the NEM is assessed at runtime after deployment at production level, while offline trust index refers to an evaluation made prior to the deployment at production level and therefore in a context different to the one in production.

- NEM instance information:
  - Static information:
    - NEM class description (manifest): features, possible actions, needed inputs, provided outputs and configuration options
    - Offline trust index: measures the trustworthiness of the behaviour of the NEM.
    - Certification level. Before being deployed in a production environment, each NEM should pass certification tests. The evaluation of those tests classifies the NEM into a given certification level.
  - Dynamic information:
    - NEM phase: current state of the NEM instance
    - Current NEM Mandate, including the actual management entities and configuration options. Historical access to previous mandates must also be provided.
    - NEM instance description, including actual inputs and outputs and possible actions

Table 8: Supervision operation

Name	**Supervision**
Description	Supervision operation facilitates the extraction of the information about the status of network elements, services, and NEMs.
Constraints	
List of input data	• Human operator requests for specific information, e.g. status of network elements and services, performance of a given NEM or group of NEMs, trust index of a given NEM or group of NEMs…

List of output data	• Alert information: ServiceProblem, ResourceProblem
	• Network monitoring information : ServiceStatisticalInfo, ResourceStatisticalInfo, Performance, ResourceStateInfo, ServiceStateInfo
	• Actions information: NEMAction
	• Deployed NEMS and characteristic information :
	  ○ Static information : NEM class description, offline trust index, certification level
	  ○ Dynamic information : phase, mandate, instance description, online trust index

List of non-functional requirements	• good graphical user interface • easy to use • information about alerts should be prioritized and filtered (no recurring information should be presented) • Most important information should be understandable at a glance. Human operator must be able to retrieve detailed information on demand

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

3.3.2 Policy Derivation and Management function

Operators need to control the behaviour of autonomic elements and set technology agnostic high level objectives that have to be properly enforced on the different elements. More importantly, operators need to customize the operation of their highly complex, heterogeneous and decentralized network on the fly. This fact necessitates the orchestrated behaviour of various entities, which may belong to different segments and operational layers. So, the policies which will be used from Governance have to span entities of different layers, administrative domains and network segments. The preservation of the relationships of policies between the respective different levels of abstraction, used and influenced by different entities/elements, necessitates policy continuum [6]. Briefly, the purpose of the Policy Continuum is to provide a semantic linkage between different types of policies that exist at different levels of abstraction. Each of the levels is optimized for a different type of user that needs and/or uses different information. For example, a business user may need SLA information. That user is not interested in how the network is programmed to deliver QoS, just that the network is in fact delivering the right type of QoS to the right people. Conversely, the network administrator may want to "translate" the QoS that is implied by different SLAs into sets of different CLI commands to program the appropriate devices. This is a completely different representation of the same policy.

Policy continuum enables policies of high level of abstraction, written based on the concepts, terminology and structure/syntax (namely, different policy model and policy language) of this level, to be transformed to policies of lower level of abstraction, with different corresponding characteristics. Policy continuum, in essence, considers a policy as a potential continuum of manifold different policies with dependencies among them. UniverSelf adopts a Policy Continuum comprised of three policy levels in parallel with Enhanced Telecom Operations Map (eTOM), which has been developed by the TeleManagement Forum [7]. This choice was based on the fact that eTOM is a business process model that provides the business-oriented view of service provider requirements that the management services and functions need to support.

The proposed policy levels are the Business level, the Service level and the NEM level (Figure 11). The Business level corresponds to "Market, product & customer" layer of the highest conceptual view of eTOM (eTOM business process framework – Level 0 processes). The Service level and NEM level corresponds to "Service" layer and "Resource (Application, computing and network)" layer, respectively. More precisely, the Business level policies are related to Strategy, Infrastructure and Product (SIP) and Operations (OPS) processes (eTOM business process framework – Level 1 processes). The Service level policies are related to Service management and operations processes of OPS, and the NEM level policies are related to Resource management and operations processes of OPS [8]. This policy hierarchy enables the definition of new sub-levels in the future, based on the emerging network and service requirements. Furthermore, the specification of policy levels based on eTOM, which entails the usage of basic common terminology, facilitates the policy translation between the different levels, enabling policy continuum.

The Business level policies, which are administration oriented and technology independent policies based on the technology agnostic business objectives and service requests, are transformed into service policies that are service oriented and technology independent, and correlate service characteristics to specific network parameters. The service level policies, in sequence, are transformed to NEM policies, which are technology specific configuration commands related to demanded operation of resources in specific network segments. The NEM policies that are imposed on the corresponding NEMs trigger specific actions from NEMs, leading to vendor specific commands enforced into specific managed network elements. Concisely, the main objective of GOV operation is the automatic transformation of business goals to NEM's configuration.

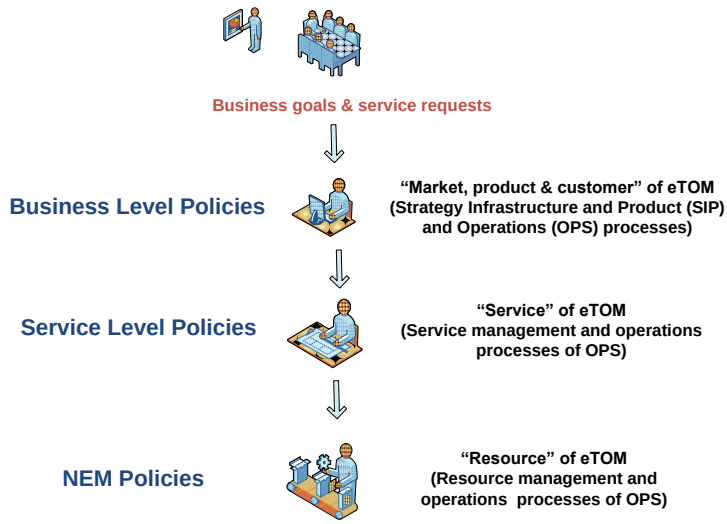


Figure 11: Schematic representation of the business, service and NEM policy levels

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 16 and Figure 17 show the representation of a policy rule and policy structure in UMF.

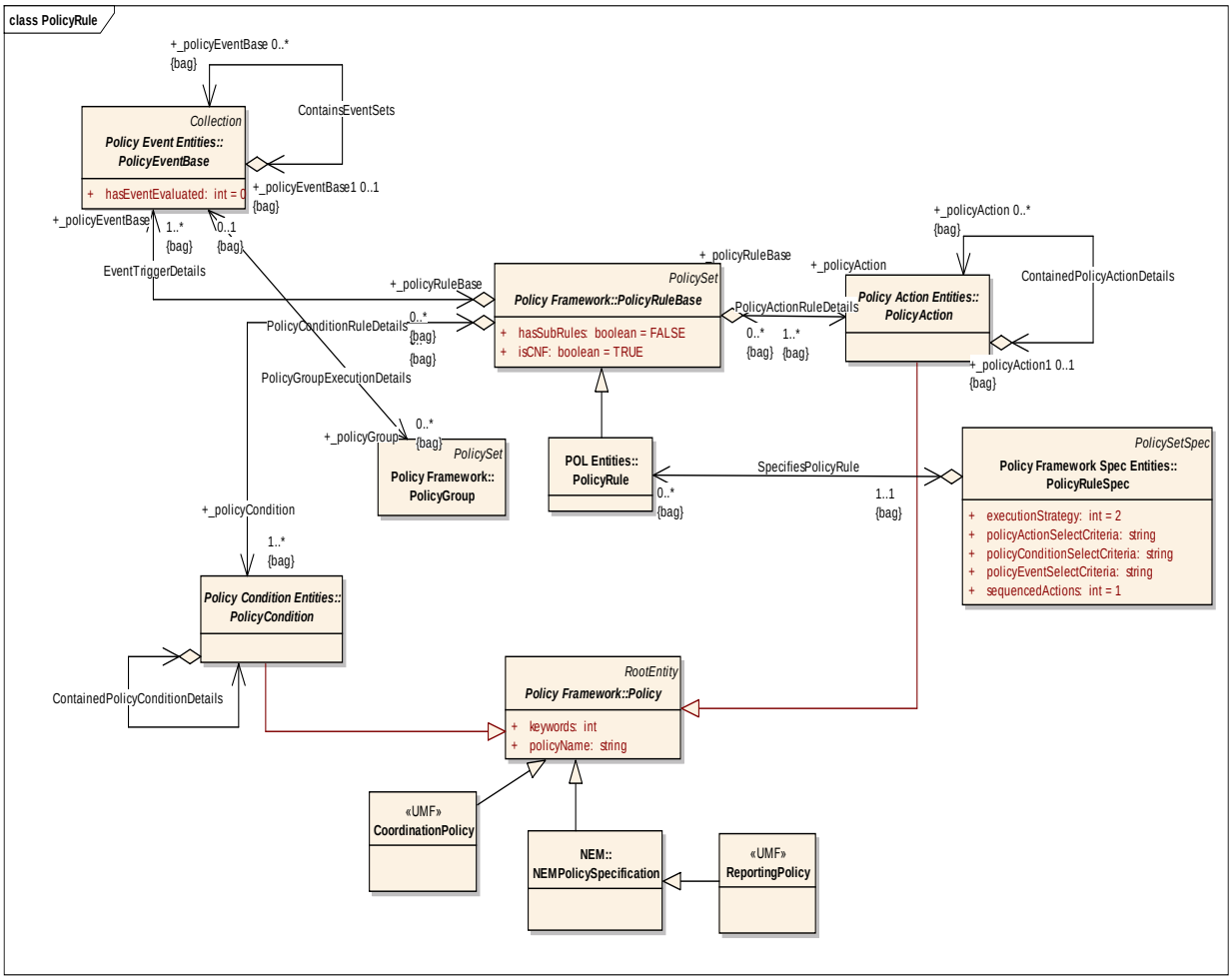


Figure 16. Representation of a PolicyRule.

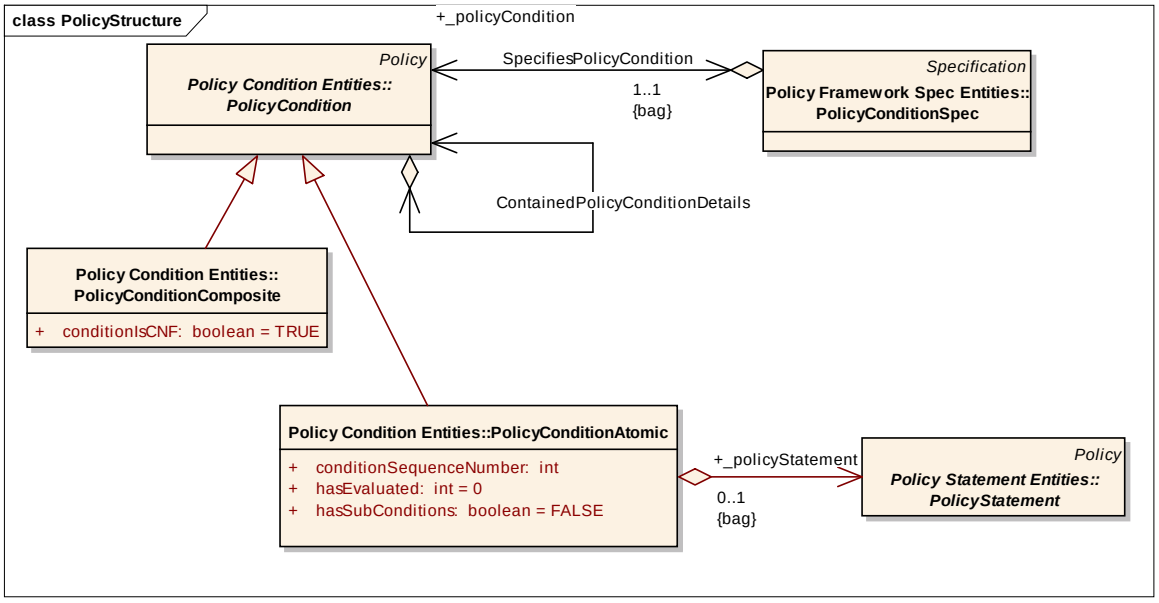


Figure 17. Representation of PolicyStrusture.

The Policy Derivation and Management (PDM) function has been decomposed in the following operations:

- Build Business Policy
- Policy Repository Management: Create/Update/Retrieve/Delete Policy operations
- Validate Policy
- Detect Policy Conflicts
- Resolve Policy Conflicts
- Translate Policy
- Check Feasibility & Optimize
- Policy Efficiency

These operations must be applied in each of the levels of the Policy Continuum, with the exception of Build Business Policy operation that only applies to the business level of the continuum. A detailed description of the above mentioned operations in terms of inputs and outputs is provided in the following subsections.

Build Business Policy

This operation transforms a High Level Objective into a policy of the upper layer of the Policy Continuum, that is, into a Business Policy. This Business Policy can then be the subject of the other operations in the same level of the continuum, and therefore this policy is stored, checked for possible conflicts with other business policies, transformed into conflict-freepolicies (in case conflicts exist), checked for feasibility and finally translated to Service Policies.

Table 9. Build Business Policy operation

Name	**Build Business Policy**
Description	Transforms high level objectives into Business Policies
Constraints	High level objectives are provided by BSS operator via a specialised human to network GUI.
List of input data	High level objectives

List of output data	Business Policies
List of non-functional requirements	N/A

Policy repository management operations

Each policy in each level of the continuum must be stored in the policy repository, and therefore operations must exist for its management: creation, update, deletion and retrieval of policies:

Table 10: Create Policy Entry operation

Name	**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	Policy, Notification (OK/NOK)
List of non-functional requirements	

Table 11: Update Policy operation

Name	**Update Policy**
Description	Update the policy content in the policy repository
Constraints	Precondition: policy exists in the repository
List of input data	Policy description/format
List of output data	Policy, Notification (OK/NOK)
List of non-functional data	

Table 12: Delete Policy operation

Name	**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	

Table 13: Retrieve Policy operation

Name	**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	

Validate Policy

This operation examines the syntax of each policy to ascertain incorrect formed policies:

Table 14: Validate Policy operation

Name	**Validate Policy**
Description	Validate the correctness (in terms of syntax and values) of a policy
Constraints	
List of input data	Policy (Business, Service, or NEM level)
List of output data	Boolean indicating whether the Policy is syntactically valid or not.
List of non-functional requirements	

Detect Policy Conflicts

Detect Policy Conflicts operation is used to examine policies for conflict at each level of the policy continuum. , A conflict is defined to occur when two policies can be triggered and satisfied at the same time and where their actions contradict each other [9].

Table 15: Detect Policy Conflicts operation

Name	**Detect Policy Conflicts**
Description	Detect conflicts between policies in the same level of the continuum
Constraints	Preconditions: more than one policy exists in the Policy repository in the same level of the continuum.
List of input data	Policy list
List of output data	Boolean, conflicted policy list
List of non-functional requirements	

Resolve Policy Conflicts

In case the Detect Policy Conflicts operation has detected a conflict between a set of policies, the Resolve Policy Conflicts operation tries to automatically resolve the conflict:

Table 16: Resolve Policy Conflicts operation

Name	**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	

Translate Policy

Business policies need to be translated or refined into lower level policies so they are expressed into a form that can be used to configure the NEMs. Therefore, at each level of the continuum a policy translation operation takes place, able to translate each policy into a policy of the next level.

Table 17: Translate Policy operation

Name	**Translate Policy**
Description	Translate business policies to service policies and then 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	

Check Feasibility & Optimize

This operation analyzes the current status of the network and the available resources. The objective is finding out which kind of optimization should be performed before the policy enforcement.

Table 18: Check Feasibility & Optimize operation

Name	**Check Feasibility & Optimize**
Description	This operation decides if some kind of optimization should be done for the network to accommodate the requests defined by the policy.
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	

Policy Efficiency

This operation measures if the network behaviour effectively follows the high level objectives set by the operator through the Human to Network function.

Table 19: Policy Efficiency operation

Name	**Policy Efficiency**
Description	Assess the successful translation of business level to Service and NEM level policies, that is, if the derived policies accomplish the goals described by the operator in the high level objectives operation. 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.
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	

3.3.3 NEM Management function

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.

This gives the human network operator the possibility of governing the autonomic behaviour of the network through the control of the operation of a particular NEM or set of NEMs. NEM Management function can be decomposed in the following operations:

- NEM Registry Management operations
- NEM Lifecycle Management operations
- Request specific behavior for NEMs operation
- Reporting Strategy-related operations

The following subsections present a detailed description of the NEM Management operations.

NEM Registry Management operations

The information related to each NEM instance should be kept in a NEM registry, and therefore operations must exist for its management: creation, update, deletion and retrieval of NEM information:

Table 20. Create NEM Entry operation

Name	**Create NEM Entry**
Description	Insert NEM INSTANCE ID into the NEM registry
Constraints	NEM registry address should be available
List of input data	NEM INSTANCE ID or List of NEM INSTANCE IDs
List of output data	Notification (OK/NOK)
List of non-functional requirements	

Table 21. Delete NEM Entry operation

Name	**Delete NEM Entry**
Description	Delete NEM entry from the NEM registry and the corresponding NEM information (NEM mandate,

	NEM instance description)
Constraints	NEM previously stored in the registry
List of input data	NEM INSTANCE ID or List of NEM INSTANCE IDs
List of output data	Notification (OK/ERROR)
List of non-functional requirements	Information of dependant NEMs should be updated

Table 22. Retrieve NEM Information operation

Name	**Retrieve NEM Information**
Description	Get NEM information from the NEM registry
Constraints	NEM previously stored in the registry
List of input data	NEM INSTANCE ID or List of NEM INSTANCE IDs
List of output data	NEM-related information: status, Manifest, NEM Instance Description
List of non-functional requirements	

Table 23. Update NEM Information operation

Name	**Update NEM Information**
Description	Updates the status or the mandate of a NEM in the NEM registry
Constraints	NEM previously stored in the registry
List of input data	NEM INSTANCE ID or list of NEM INSTANCE ID
List of output data	Notification (OK/ ERROR)
List of non-functional requirements	

NEM Lifecycle Management operations

A NEM from the moment that it is installed until the moment that it is uninstalled follows the life-cycle depicted in section 3.2.1. Operations are needed to manage this lifecycle:

Table 24. Create NEM Instance operation

Name	**Create NEM Instance**
Description	Create a NEM instance in the network elements in which the NEM software is stored
Constraints	NEM software stored in the network element or a proxy
List of input data	NEM class Identifier
List of output data	Notification (OK/NOK)

Table 25. Set Up a NEM operation

Name	**Set Up a NEM**
Description	Activates a NEM to start operating
Constraints	The NEM must be on the "READY" state
List of input data	
List of output data	The result of the operation (OK/NEM_NOT_READY)

Table 26. Generate NEM Mandate operation

Name	Generate NEM Mandate
Description	Generates a new Mandate for a given NEM. This operation triggers the deployment or modifies the current deployment of a NEM. This operation retrieves the mandate from the NEM registry and embeds into it the list of equipment and policies.
Constraints	
List of input data	NEM ID, List of equipments over which deploying, Optionally NEMSpecificPolicies (e.g. optimization target)
List of output data	Mandate
List of non-functional requirements	

Table 27. Set Down a NEM operation

Name	Set Down a NEM
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	The result of the operation (OK/OK_WITH_WARNING)

Request specific behaviour for NEMs

Request specific behavior for NEMs is an operation which permits GOV to request specific control of NEMs, in order to modify the mode of operation of a given NEM or group of NEMs. This feature allows for instance the network operator to set NEMs into the so-called “under trial” mode, where NEMs are not allowed to enforce actions on the network. The behavior of the NEM can then be observed without risking the network operation.

There are two immediate usages for this functionality: observation of a misbehaving NEM and certification activities. In the former case, when the online trust mechanisms find a deviation of the runtime behavior of a NEM with respect to the specified behavior, the human network operator may decide, instead of stopping the NEM, the deactivation of any kind of actions of the NEM on the network and still observe its activity. Alternatively, instead of manually requesting the modification of the NEM behavior, the human operator may use the H2N function to set a threshold for the online trust index. When the computed trust index for a given NEM drops below the threshold, the trust mechanism in the KNOW block issues a Call For Governance[7], which triggers the NEM Management function in order to set the NEM in the “under trial” state. Similar observational needs may appear when a new NEM is deployed during a certification process or at the early stages of its

[7] A Call for Governance is issued in those cases where a problem occurs and can not be solved in the appropriate element (NEM or UMF core block). The ultimate goal of this notification is to scale up the problem to the human network operator. The example mentioned in the text corresponds to a situation when a deployed NEM is found not to be trustworthy. A second example that will appear later in the text corresponds to a situation when a NEM finds that it can not perform correctly in its current context. In that case, a notification is sent to COORD, that will check if some of the coordination mechanisms have failed. If this is not the case, then COORD will issue a Call for Governance to GOV core block.

instantiation at production level, allowing the examination of the NEM activity and its process of decision making without allowing the actual execution of corrective or preventive actions on the network elements.

Given than a NEM in OPERATIONAL state runs under the control of the COORD core block, the communication between both blocks is fundamental in this case, so COORD can take appropriate actions to guarantee the stability of the network. Take for instance the case when the NEM to be set "under trial" is part of a group of NEMs being orchestrated. The fact that one of them will be set to the "under trial[8]" state may affect the behavior of the group and therefore COORD may need to send new control policies to those NEMs. In order to ensure the proper coordination of the two blocks, GOV will not communicate directly with the NEMs, but will request COORD to control a given NEM. This request will be in the form of policies, which will be enforced through the invocation of Send COORD Policies operation of the Enforcement function. As part of these policies, GOV may request different options to COORD. Actually, it may request a change in COORD mechanism settings, in order to see the behavior of the NEM as if it was acting and which would be the influences on the other NEMs, or it may just wish to observe the NEM working in isolation of other NEMs.

Table 28. Request specific behaviour of NEMs operation

Name	**Request specific behavior for NEMs**
Description	Modifies the working behaviour of a given NEM, so it works under a specified operational mode
Constraints	The NEM must be in "READY" or "OPERATIONAL" state.
List of input data	New NEM specific behaviour
List of output data	The result of the operation (OK/ERROR)

Reporting Strategy-related operations

NEM reporting strategy operations allow the human operator to define the frequency and degree of detail of the information the NEM must report to the UMF core. This operation may be useful under specific circumstances when the behavior of a NEM must be closely observed (e.g. when a malfunctioning has been detected, or when an event has been programmed that will increase the demand of services in a given area). In that case, the human network operator may wish to increase the frequency and details of the reported information. The Trust mechanisms in KNOW block also needs to set configuration options to NEMs about the reporting information. This includes:

- The frequency of the NEM reporting of information
- Type of reporting: batch/real time
- Type of reported information: inform about all the decisions taken or only about a sample of them

Table 29: Build Reporting Strategy operation

Name	**Build Reporting Strategy**
Description	Build reporting strategy composed by several reporting policies
Constraints	
List of input data	Reporting policies
List of output data	Reporting Strategy Set<ReportingPolicy>
List of non-functional requirements	

Table 30: Send Reporting Strategy operation

[8] see Figure 13. Information model of NEM regimes

Name	Send Reporting Strategy
Description	Send reporting strategy to the NEM
Constraints	
List of input data	List of PolicySet, ID or list of INSTANCE IDs
List of output data	the result of the operation (OK/NOK)
List of non-functional requirements	

3.3.4 Enforcement

Enforcement encapsulates the communication mechanism between Governance and NEMs or other UMF core components. It allows the other functions of the Governance block to be independent of the communication aspects for the interconnection with NEMs and core blocks. In the case of the NEMs, the communication is mainly achieved through the MANDATE object. The following tables summarize the operations of the Enforcement function:

Table 31. Send NEM Mandate operation

Name	Send NEM Mandate
Description	Sends a new Mandate to a given NEM. This operation is used to set or modify the deployment and the settings of a NEM.
Constraints	
List of input data	Mandate
List of output data	Notification (OK/NOK)
List of non-functional requirements	

Table 32. Send COORD Policies operation

Name	Send NEM Specific Policies
Description	Sends new policies to modify the NEM algorithm settings, the structure of these policies are obtained from the NEM Manifest analysis.
Constraints	
List of input data	List of policies, NEM instance ID
List of output data	Notification (OK/NOK/REQUIRES_REDEPLOY), it may happen that the actual content of the policy requires the NEM to modify its deployment, hence if the NEM is in the “OPERATIONAL” state, then GOV will need first to SetDown this NEM.
List of non-functional requirements	

Table 33. Send COORD Policies operation

Name	Send COORD Policies
Description	Sends SpecificCOORDPolicies to govern the behaviour of the Coordination block. These policies define the conflict types to be addressed by COORD, the sensitivity of the COORD functions, and the

	orientation to be used when prioritizing the NEMS.
Constraints	
List of input data	List of policies
List of output data	Notification (OK/NOK)
List of non-functional requirements	

Table 34. Send KNOW Policies operation

Name	**Send KNOW Policies**
Description	Sends new policies to govern the behaviour of the KNOW block. These policies are OptimizationGoalPolicies or TrustPolicies.
Constraints	
List of input data	List of policies
List of output data	Notification (OK/NOK)
List of non-functional requirements	

The following sequence activity diagrams illustrate GOV operation and interaction with the other UMF core blocks.

In the NEM policy definition activity diagram in Figure 18, the Business Operator sets high level objectives representing a set of goals that the network should meet (operation: ***High Level Objectives Definition***). These high-level objectives are transformed into Business Policies (operation: ***Build Business Policy***). GOV checks the correctness of the policy (*operation:* ***Validate policy***), check for potential conflicts (operation: ***Detect Policy Conflicts***), and, in case of conflicts, tries to solve it (operation: ***Resolve Policy Conflicts***). Then, Business policies are translated into Service policies (*operation:* ***Translate policy***). GOV then controls the correctness of the Service policy (*operation:* ***Validate policy***), checks and resolves conflicts (operations: ***Detect Policy Conflicts*** and ***Resolve Policy Conflicts***), and assesses the feasibility of the new Service policy (operation: ***Check Feasibility & Optimize***). This assessment operation needs information about the status of the network and the available resources, and therefore the KNOW block is invoked (**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 into NEM policy (*operation:* ***Translate policy***). After the control of NEM policy correctness (*operation:* ***Validate policy***), and the checking and potential conflicts resolution (operations: **Detect Policy Conflicts** and **Resolve Policy Conflicts**), GOV checks again the feasibility of the policy (operation: ***Check Feasibility & Optimize***). This implies again the invocation of the KNOW block, requesting 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 outcome of the policy derivation activity is a list of NEM policies. In case an error occurs during the execution of the activity, a notification is sent to the human operator through the H2N function (operation: **Supervision**).

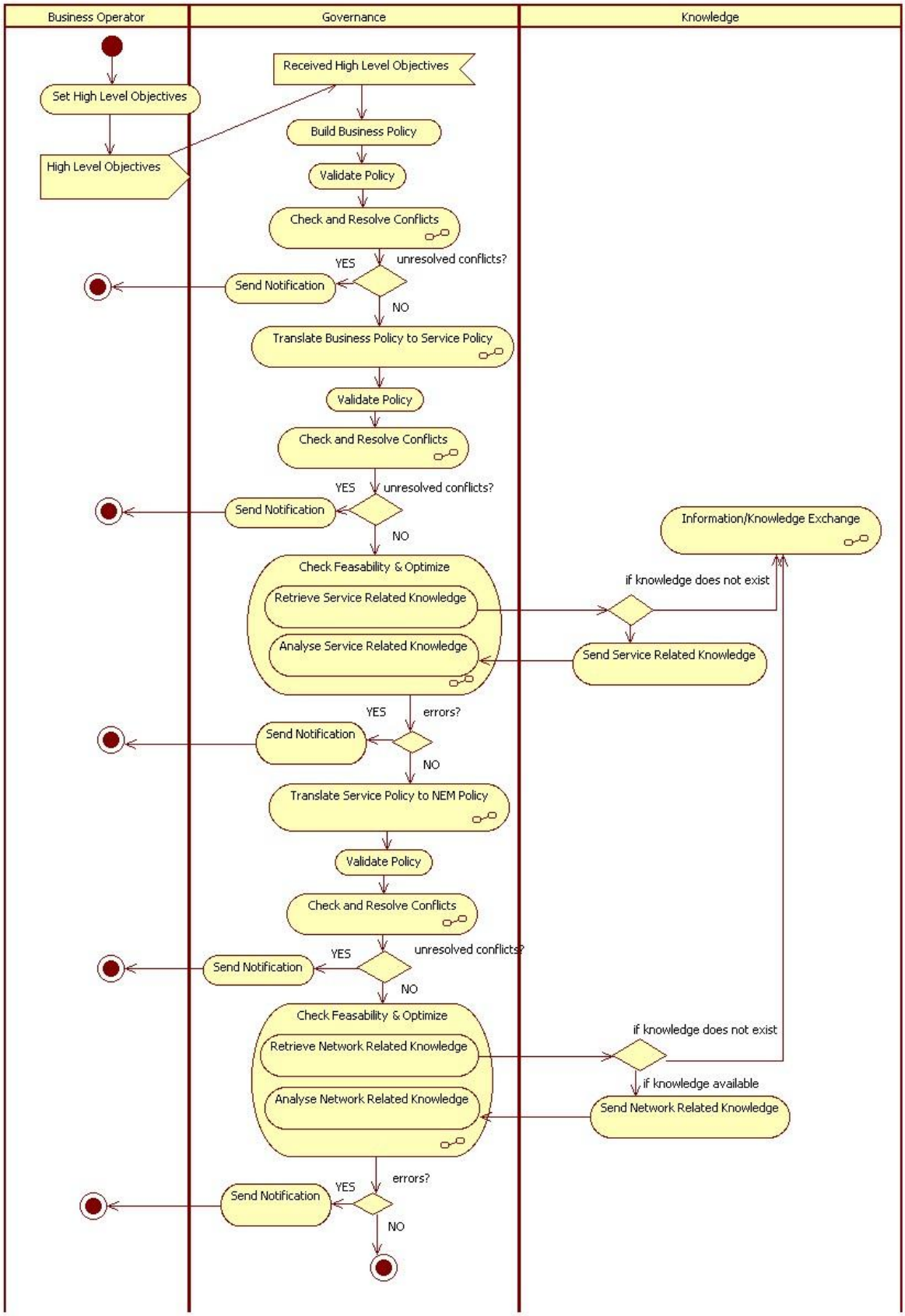


Figure 18. NEM policy definition activity diagram.

The following figure presents the activity diagram corresponding to the NEM instantiation activity:

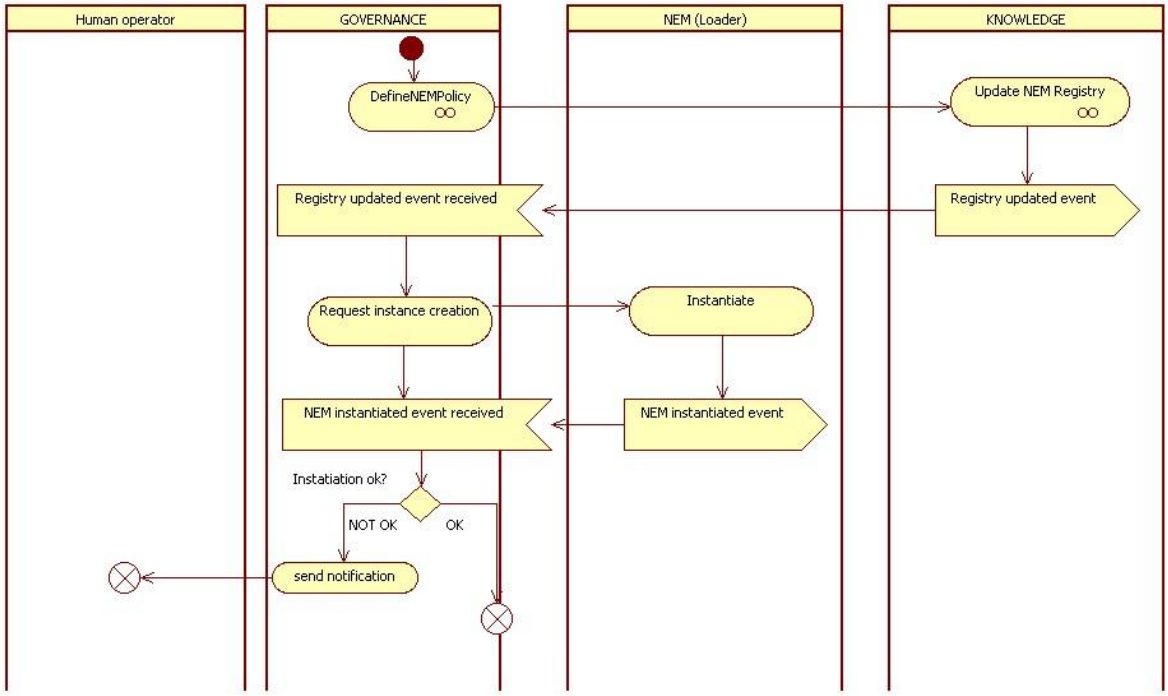


Figure 19. NEM instantiation activity diagram

For a NEM policy to be deployed and enforced, the corresponding NEMs must have been already instantiated in the network. Figure 19 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 (operation: ***Create NEM Instance***) sends to the corresponding NEM loader the instruction to create an instance (**GOV-NEM interface**). A notification is sent back to the GOV block to report on the action, and showed to the human operator through the H2N function (operation: **Supervision**). 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 20Figure 20 illustrates the activity diagram of the NEM configuration implemented through the enforcement of a new Mandate. When the human operator needs to change the deployment options of a NEM, it examines if the corresponding NEM is in READY state. If it is not, GOV sets down the NEM (operation***: Set Down a NEM***) to bring it in a READY state. When GOV accomplishes this procedure or if the NEM is in READY state, then GOV creates a new NEM Mandate (operation: ***Generate NEM Mandate***) and sends (operation***: Send NEM mandate***) it to NEM (***GOV-NEM interface***). In case there is any problem that 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 (***GOV-NEM interface***), which makes it available to the human operator through the H2N function (operation: **Supervision**). If the NEM deployed the new demanded status, then GOV starts up the NEM (operation***: Set up a NEM***).

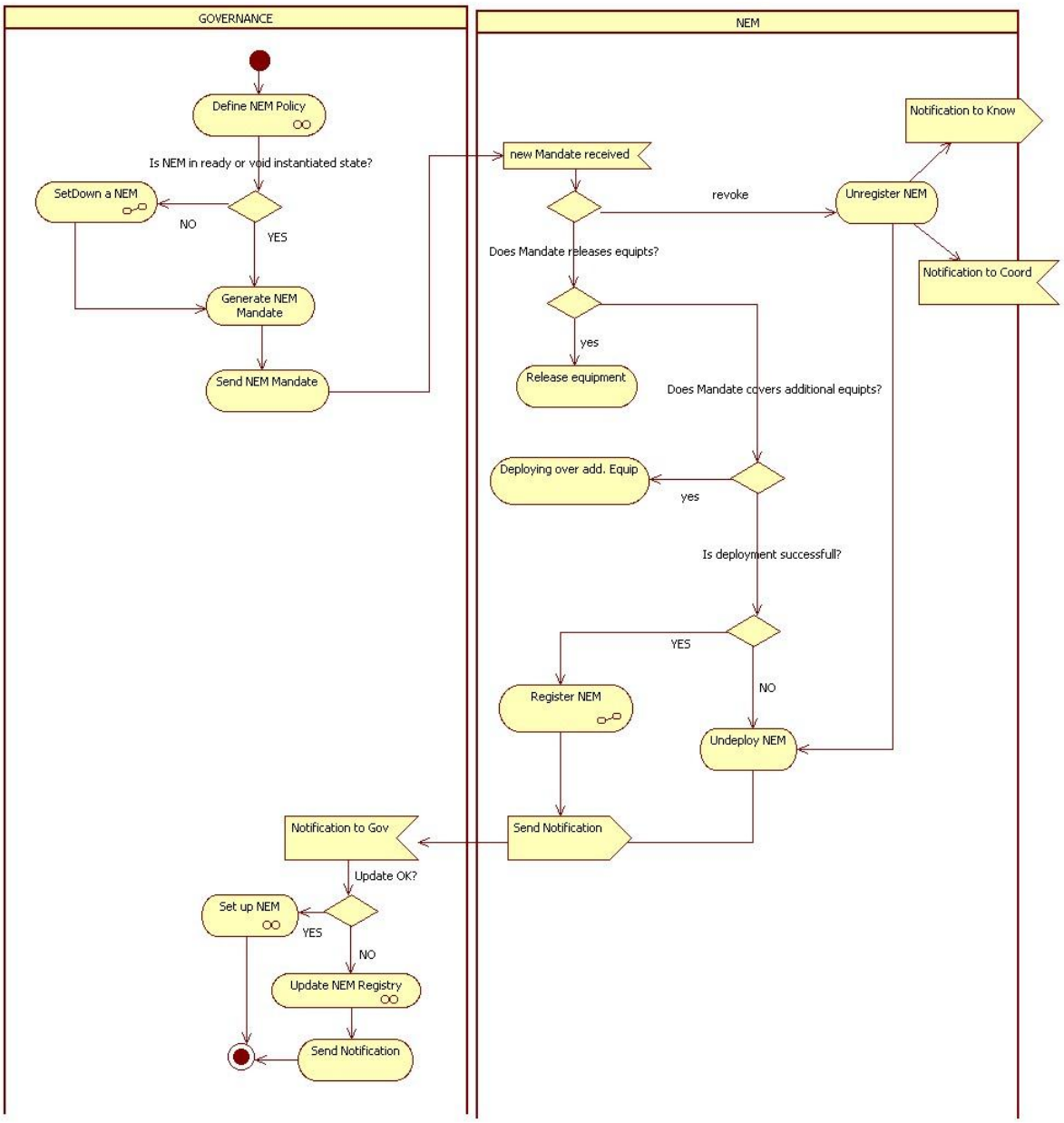


Figure 20. Update Mandate activity diagram.

Figure 21 illustrates the activity diagram of the GOV activating or deactivating a NEM.

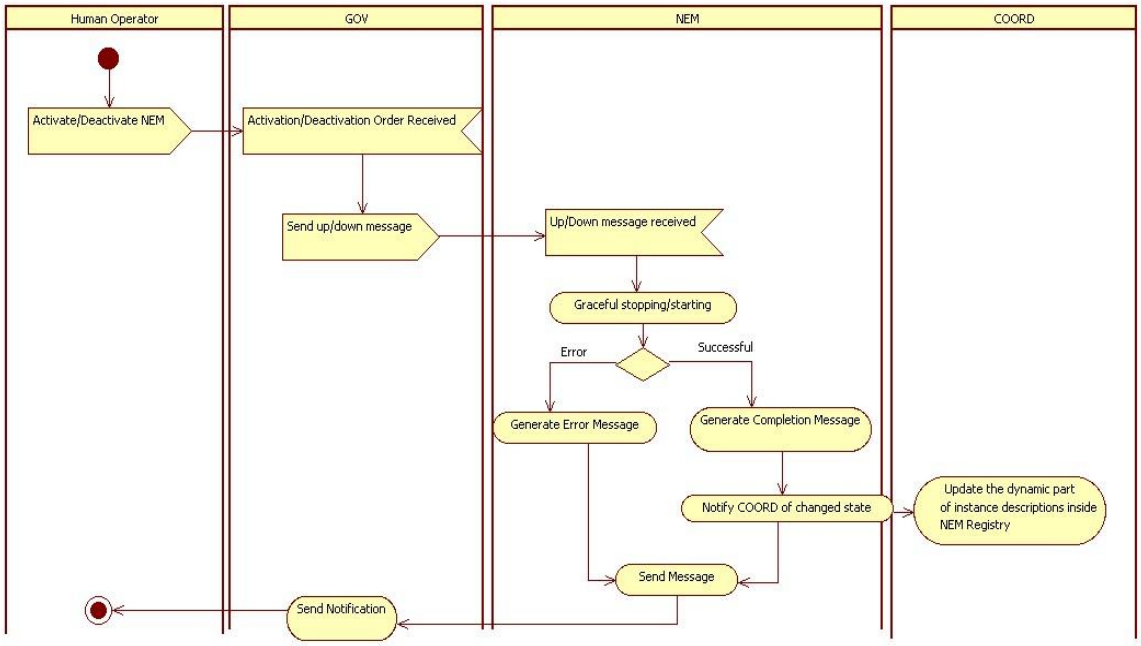


Figure 21. Change NEM operational state diagram

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

Figure 22 depicts the registration phase of a NEM, which just deployed after receiving a Mandate. The NEM is sending its instance description 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 3.2.5 and 3.2.6 for a description of the Mandate and Instance Description).

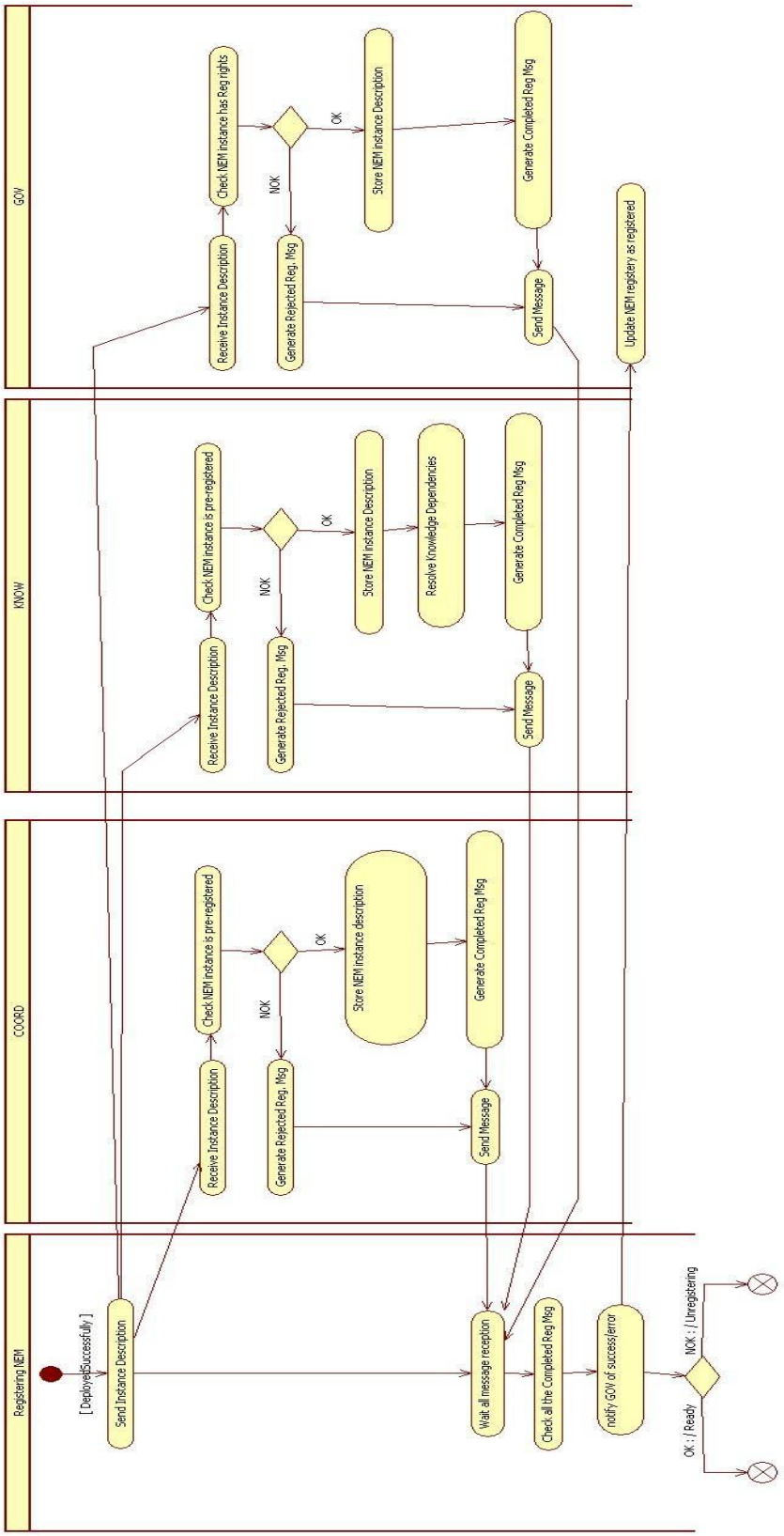


Figure 22. Register NEM activity diagram

3.4 Knowledge block

The UMF Knowledge block (KNOW) 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

The Knowledge Block offers basic information/knowledge manipulation functionalities to the UMF entities through the ***Knowledge Exchange Interface***. The ***Knowledge Management Interface*** handles information flow management that includes configuration actions towards the optimal handling of the information/knowledge in the UMF. A more detailed description of the interfaces can be found below.

3.4.1 KNOW machine readable description

Here below is provided an indicative format of the KNOW machine readable description, which GOV uses in order to determine which are the configuration options applicable to the KNOW implementation available in the UMF system.

```
<?xml version="1.0"?>
<eu.univerself.nem.Manifest>
   <ConfigurationOptions>
      <SpecificKNOWPolicy>
         <name>Set KNOW Global Optimization Goal to Reduce Average Response
Time</name>
         <description>This policy allows GOV to enforce a global optimization goal
that reduces the average response time of all established information flows. KNOW
enforces the optimization goal, through:
                  - Placing extra KNOW nodes supporting the ICD function close to
               the entities communicating information
                  - Using KNOW nodes that support many or all of the KNOW
               functions (i.e., to avoid distribution of information to remote KNOW
               nodes).
                  - Using storage technologies that support information caching.
                  - Using responsive knowledge production mechanisms.
                  - Applying relaxed information accuracy objectives in order to
               minimize communication cost.
                  - Using information dissemination mechanisms that minimize
               response time (e.g., prefer push to pull).
                  - Prefering direct NEM2NEM communication to communication
               through KNOW.
         </description>
         <parameters>
            <description>Extra optimization parameters that may be
         defined</description>
         </parameters>
         <LevelOfEnforcement>{Low, Medium, Heigh}
            <defaultValue>High</defaultValue>
            <description>The level of goal enforcement. For example, high level
               defines a critical situation where the enforcement should override
               all other relevant decisions been taken.
            </description>
         </LevelOfEnforcement>
         <ReportingCapabilities>
            <KPIs>{Response Time, Communication Overhead, Information Accuracy}
            </KPIs>
            <description>KPIs that can be measured for the specific global
         performance optimization goal and communicated to the GOV.
            </description>
         </ReportingCapabilities>
      </SpecificKNOWPolicy>
      <SpecificKNOWPolicy>
```

```
      <name>Set KNOW Global Optimization Goal to Reduce KNOW Processing
   Cost</name>
      <description>This policy allows GOV to enforce a global
      optimization goal that reduces the KNOW processing cost. KNOW
   enforces the optimization goal, through:
                - Using less KNOW nodes
                - Using lightweight knowledge production mechanisms.
                - Applying relaxed information accuracy objectives in order to
            minimize the need for processing.
      </description>
      <parameters>
         <description>Extra optimization parameters that may be
      defined</description>
         </parameters>
      <LevelOfEnforcement>{Low, Medium, Heigh}
         <defaultValue>High</defaultValue>
         <description>The level of goal enforcement. For example, high
      level defines a critical situation where the enforcement
   should override all other relevant decisions been taken.
         </description>
      </LevelOfEnforcement>
      <ReportingCapabilities>
         <KPIs>{Processing Cost, Information Accuracy}</KPIs>
         <description>KPIs that can be measured for the specific global
      performance optimization goal and communicated to the GOV.
         </description>
      </ReportingCapabilities>
   </SpecificKNOWPolicy>
   <SpecificKNOWPolicy>
      <name>Set KNOW Global Optimization Goal to Reduce
      Information/Knowledge Handling Communication Overhead</name>
      <description>This policy allows GOV to enforce a global
      optimization goal that reduces the average communication
   overhead of all established information flows. KNOW enforces                the
optimization goal, through:
                - Placing extra KNOW nodes supporting the ICD function close to
            the entities communicating information
                - Using KNOW nodes that support many or all of the KNOW functions
            (i.e., to avoid distribution of information to remote KNOW nodes).
                - Using storage technologies that support information caching.
                - Applying relaxed information accuracy objectives in order to
            minimize communication cost.
                - Prefering direct NEM2NEM communication to communication through
            KNOW.
      </description>
      <parameters>
         <description>Extra optimization parameters that may be
      defined</description>
      </parameters>
      <LevelOfEnforcement>{Low, Medium, Heigh}
         <defaultValue>High</defaultValue>
         <description>The level of goal enforcement. For example,
      high level defines a critical situation where the enforcement
   should override all other relevant decisions been taken.
         </description>
      </LevelOfEnforcement>
      <ReportingCapabilities>
         <KPIs>{Communication Overhead, Information Accuracy}</KPIs>
         <description>KPIs that can be measured for the specific global
      performance optimization goal and communicated to the GOV.
         </description>
      </ReportingCapabilities>
   </SpecificKNOWPolicy>
   <SpecificKNOWPolicy>
      <name>Set KNOW Global Optimization Goal to Improve Information
      Accuracy</name>
```

```
         <description>This policy allows GOV to enforce a global
      optimization goal that improves the information
   accuracy of all active information flows. KNOW enforces                    the
optimization goal, through:
                  - Using accurate knowledge production mechanisms.
                  - Applying stringent information accuracy objectives.
                  - Prefering communication through KNOW, in order to exploit easier
               all available information.
                  - Disabling storage caching.
         </description>
         <parameters>
            <description>Extra optimization parameters that may be
         defined</description>
         </parameters>
         <LevelOfEnforcement>{Low, Medium, Heigh}
            <defaultValue>High</defaultValue>
            <description>The level of goal enforcement. For   example, high
            level defines a critical situation where the enforcement
      should override all other relevant decisions been taken.
            </description>
         </LevelOfEnforcement>
         <ReportingCapabilities>
            <KPIs>{Information Accuracy}</KPIs>
            <description>KPIs that can be measured for the specific global
         performance optimization goal and communicated to GOV.
   </description>
         </ReportingCapabilities>
      </SpecificKNOWPolicy>
      <SpecificKNOWPolicy>
         <name>Set KNOW Global Optimization Goal to Improve Energy
      Efficiency</name>
         <description>This policy allows GOV to enforce a global
      optimization goal that reduces energy consumption of the KNOW
   infrastructure and the information manipulation capabilities of            the
UMF entities (i.e., NEMs and core blocks). KNOW enforces the
   optimization goal, through:
                  - Placing the optimum KNOW nodes supporting the ICD function close
               to the entities communicating information
                  - Using KNOW nodes that support many or all of the KNOW functions
               (i.e., to avoid distribution of information to remote KNOW nodes).
                  - Using energy efficient storage technologies that support
               information caching.
                  - Using responsive and lightweight knowledge production mechanisms.
                  - Applying relaxed information accuracy objectives in order to
               minimize communication cost.
                  - Using energy efficient information collection / dissemination
               mechanisms.
                  - Prefering direct NEM2NEM communication to communication through
               KNOW.
         </description>
         <parameters>
            <description>Extra optimization parameters that may be defined
            </description>
         </parameters>
         <LevelOfEnforcement>{Low, Medium, Heigh}
            <defaultValue>High</defaultValue>
            <description>The level of goal enforcement. For example, high
         level defines a critical situation where the enforcement
   should override all other relevant decisions been taken.
            </description>
         </LevelOfEnforcement>
         <ReportingCapabilities>
            <KPIs>{Response Time, Communication Overhead, Processing Cost,
         Energy Consumption, Information Accuracy}</KPIs>
            <description>KPIs that can be measured for the specific global
         performance optimization goal and communicated to the GOV.
            </description>
```

```
        </ReportingCapabilities>
      </SpecificKNOWPolicy>
   </ConfigurationOptions>
</eu.univerself.nem.Manifest>
```

3.4.2 Information model of Knowledge

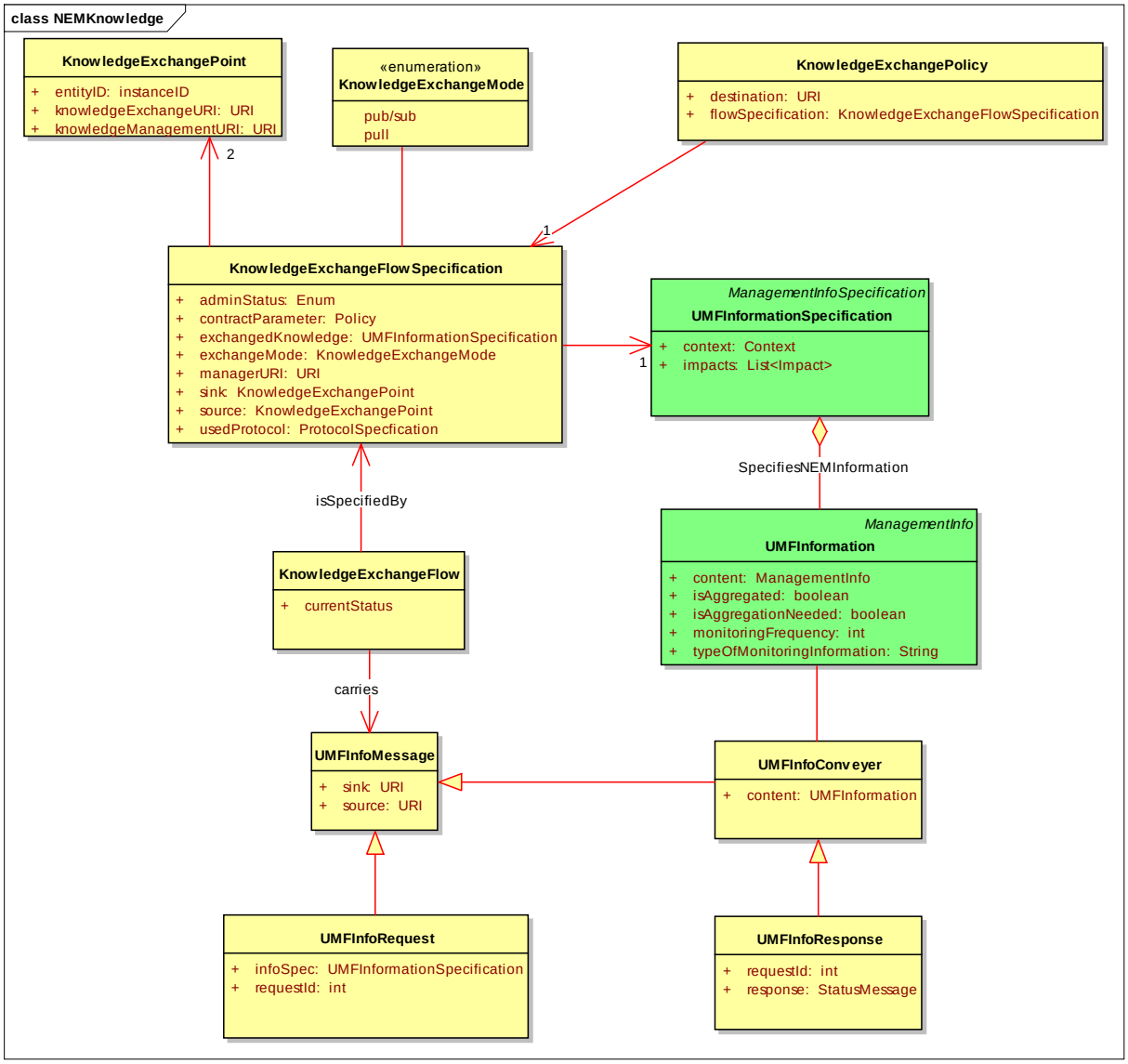


Figure 23. Information model of Knowledge

The above information model describes how the information flows are manipulated inside the knowledge block and by the Knowledge Management Interfaces (see section 3.6.2).

The most important concept is the knowledge exchange flow, which is being set between two knowledge exchange interfaces. It is specified by the two endpoints of the flow and by the UMF information Spec that is being conveyed over the flow. Some additional parameters define the way the information are being exchanged (Push or Pull mode plus additional parameters to determine the frequency and conditions of the actual information exchange).

KnowledgeExchangeFlowSpecifications are being negotiated by Knowledge Management Interfaces and the KNOW, in order to determine the characteristics of the exchange flow, which will then be enforced between two knowledge exchange interfaces (see Figure 36).

The model is also defining the fields of the messages insuring the information exchange over the Knowledge Exchange Flow.

3.4.3 Information Collection & Dissemination function

The Information Collection and Dissemination (ICD) function is responsible for information collection, sharing, retrieval and dissemination. An overview of the ICD function is shown in Figure 24.

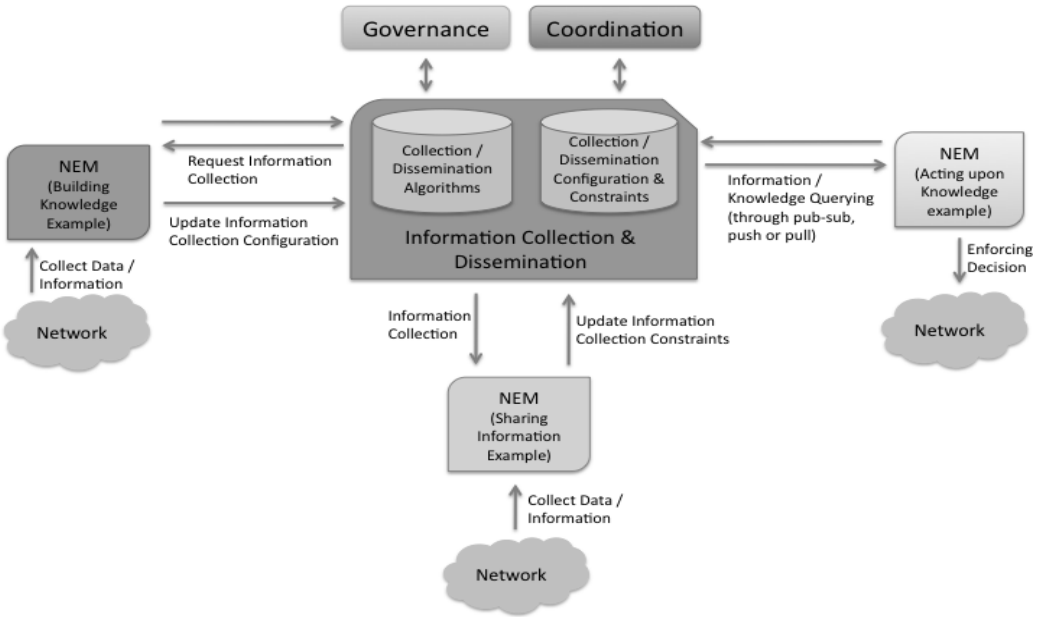


Figure 24. Overview of the Information Collection and Dissemination Function.

The KNOW block handles information from/to the NEMs and the other core blocks, 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 COORD core blocks and the NEMs.

The UMF functional components can act as sources or sinks of information. The sources subscribe to the ICD by exposing the type of information they can produce (UMFinformationSpecification). 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) 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.

Information can be directly retrieved from or shared with the KNOW block from a UMF entity using the appropriate interface (i.e., the Knowledge Exchange Interface). Furthermore, an information collection process is triggered by 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 (in case it is stored), or it may have to request a suitable Knowledge building NEM to provide such an information 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 or knowledge production) and then optionally stored/indexed to the Information Storage and Indexing function. The storage option 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.

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 by the KNOW block, in response of an information retrieval request, in case the requested information is not available in the storage. Information updates without such request are considered as information publishing processes.
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, i.e., enabling the NEMs as information sources.
List of input data	The NEMs producing the information, UMFInformationSpecification.
List of output data	UMFInformation
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 by 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, i.e., enabling the NEMs as information sources.
List of input data	UMFInformationSpecifications, UMFInformation
List of output data	ACK or NoACK
List of non-functional requirements	N/A

Operation 3	**Information retrieval**
Description	Information can be queried from UMF entities e.g. the Knowledge Building NEMs during their knowledge building process and both information & knowledge can be queried from 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	UMFInformationSpecifications
List of output data	UMFInformation
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 dissemination comes after an information retrieval request. 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 ICD function 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 37 • **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 38.
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	UMFInformationSpecification, dissemination method
List of output data	UMFInformation
List of non-functional data	N/A

3.4.4 Information Storage & Indexing function

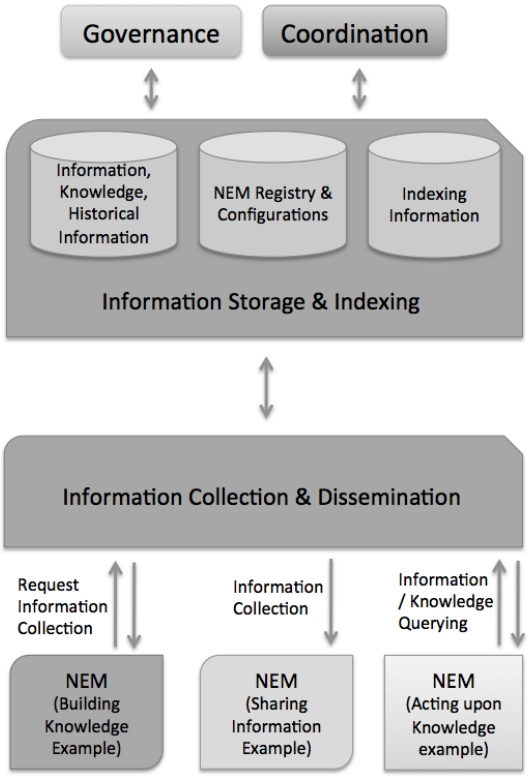


Figure 25. Overview of the Information Storage and Indexing Function.

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 25. The ISI function stores information, such as NEM registration information and 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 extended-scoped knowledge) or Knowledge Building NEMs (for locally-scoped knowledge). The different UMF entities[9] 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.

Furthermore, ISI supports: (i) pub/sub information dissemination capabilities, (ii) alternative storage structures (i.e., centralized vs distributed or hierarchical) and database technologies based on the context, and (iii) information and knowledge caching.

[9] NEM instances or core mechanisms

List of "Information Storage and Indexing" operations:

Information Storage, Information Indexing, NEM registration

Operation 1	**Information Storage**
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 UMF entity (either a NEM or a Core mechanism) through the ICD.
Constraints	
List of input data	List<UMFInformation>
List of output data	ACK
List of non-functional requirements	N/A

Operation 2	**Information Indexing**
Description	Information communicated through the ICD function is indexed. After this stage, the information could be retrieved and passed back to the ICD function for dissemination. Indexed information comes from a NEM registering itself or from a core block communicating the shareable information.
Constraints	
List of input data	List<UMFInformationSpecification> (see Figure 11 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	**NEM Registration**
Description	All NEMs should be registered to the KNOW block. This process includes registering 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 3.2.6)
List of output data	ACK
List of non-functional requirements	N/A

3.4.5 Information Processing & Knowledge Production function

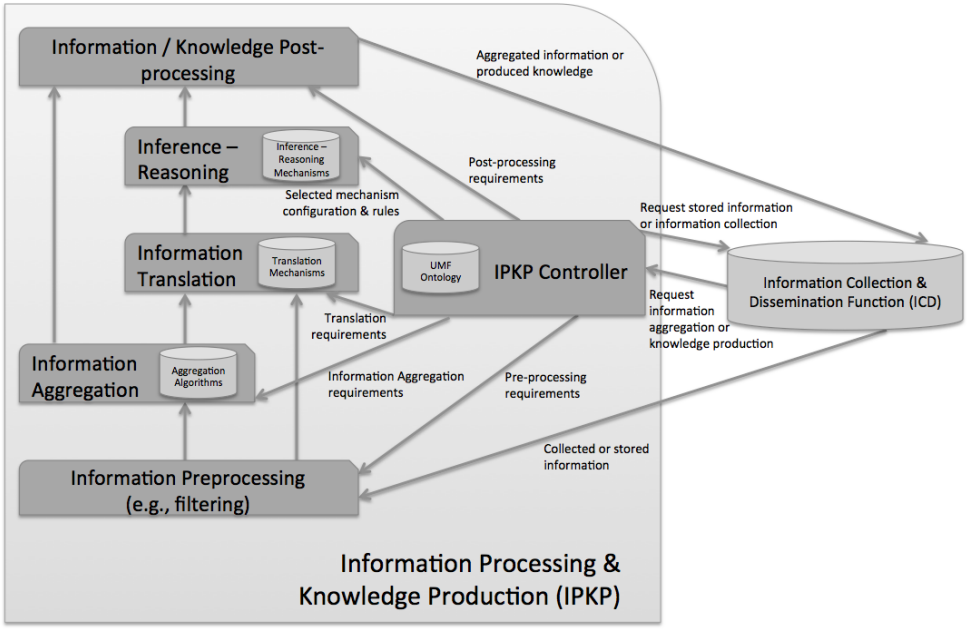


Figure 26. Overview of the Information Processing and Knowledge Production (IPKP) function.

The Information Processing and Knowledge Production function (IPKP) is responsible for operations related to information processing (i.e., aggregation) and knowledge production. The IPKP provides to NEMs and core block the necessary toolkit to produce different information abstractions, including processed information and extended-scoped knowledge.

The Knowledge Production (KP) operation handles and produces knowledge that may be extended-scoped. The latter type of knowledge is being produced out of aggregated information or locally-scoped knowledge. Locally-scoped knowledge can be built from the Knowledge Building NEMs out of data/information directly collected from the managed entities, i.e., its scope is limited to those entities. In all cases of knowledge production, reasoning and inference mechanisms are required. These mechanisms are based on different techniques depending on the exact problem addressed, the type of inputs used and the type of output that needs to be acquired. Such techniques come from scientific areas like statistics, clustering, reasoning, Fuzzy or machine learning (including supervised, unsupervised and reinforcement learning techniques). All the above information (e.g., problem addressed, type of inputs / outputs, inference/reasoning mechanisms etc) is described in the UMF ontology, ready to be looked up from the IPKP function when such a demand appears.

In Figure 26. Overview of the Information Processing and Knowledge Production (IPKP) function., we give an overview of the IPKP function and its basic operations. A NEM or UMF core service that requires the KNOW IPKP functionalities requests to utilize either an Information Aggregation (IA) or a Knowledge Production (KP) operation. The ICD function handles the communication of the UMF component with the internal IPKP functionalities and the IPKP controller is responsible to control the internal IPKP components. The two IPKP operations (i.e., information aggregation & knowledge production) require a number of basic steps:

Step 1: Determining the information aggregation or knowledge production parameters (e.g., information filtering configuration, the inference/reasoning algorithm to use, translation requirements, whether aggregation is required and/or information/knowledge post-processing requirements). This process is being handled from the IPKP controller, which matches the UMF component's requirements and the type of problem to solve with the relevant information. The parameters are being communicated to all relevant internal IPKP components.

Step 2: Collection of input information either from a UMF component that produces it or from the KNOW ISI function (i.e., the KNOW storage). A collection request is being passed back from the IPKP controller to the ICD function. This phase matches the knowledge exchange workflow detailed in section 3.6.2, where the information sink is the KNOW itself.

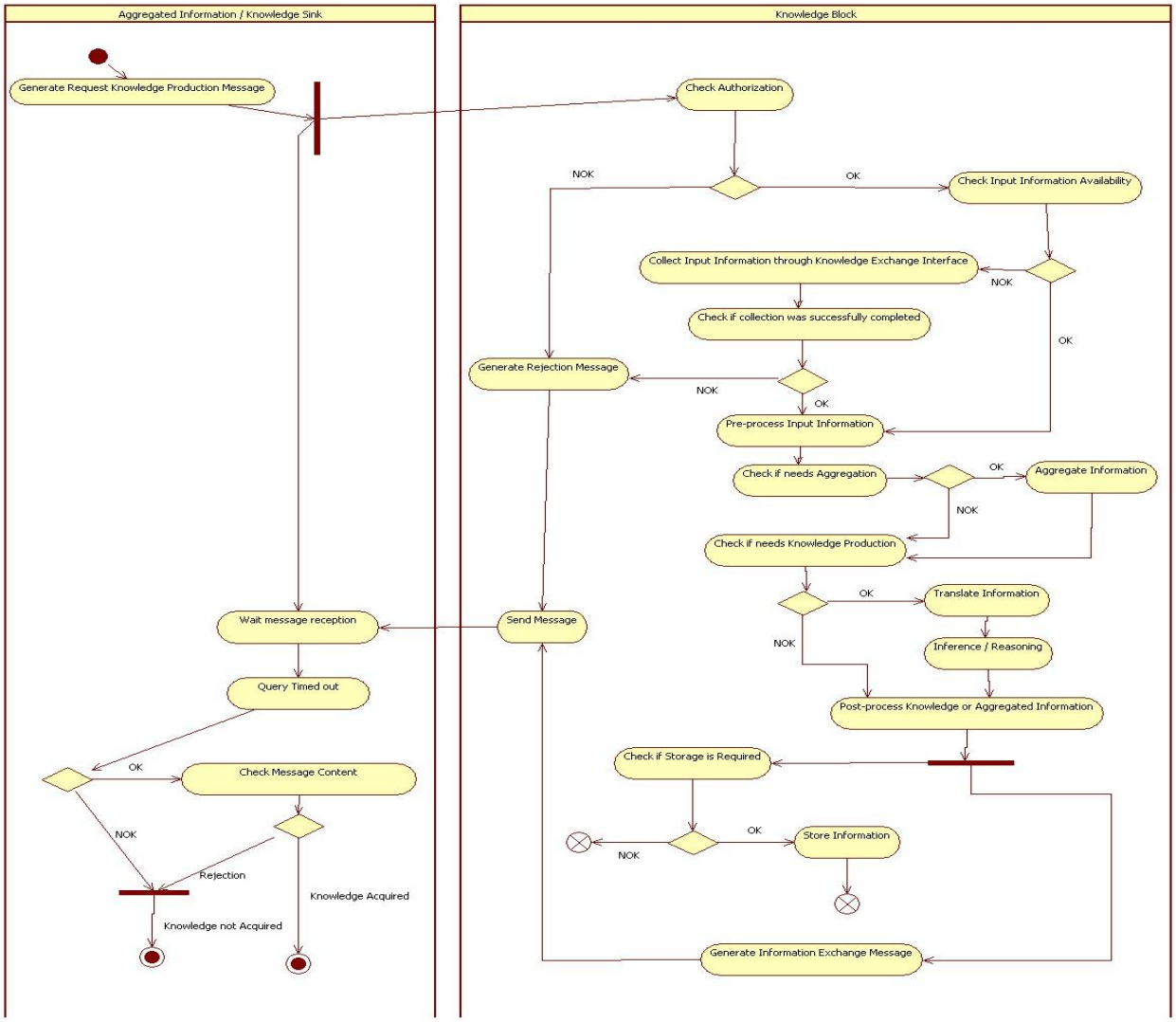


Figure 27. Overview of the Knowledge Production Workflow

Step 3: Pre-processing of the input information (e.g., applying information filtering) that may be required. The pre-processing requirements are being set from the IPKP controller.

Step 4: The input information is being passed to the IA operation in case of information aggregation, where an aggregation process takes place according the requirements (e.g., aggregation function used) being set from the IPKP controller. In case of knowledge production, this step may be bypassed or not (i.e., the higher-level knowledge production processes may require aggregation before the inference/reasoning process).

Step 5: In case of knowledge production, the input information may need to be translated in a convenient representation, e.g., to OWL. The translation configuration is being set from the IPKP controller to match the requirements of the inference/reasoning mechanism identified from the UMF ontology.

Step 6: The actual inference/reasoning process takes place in this step. The input information (i.e., in an appropriate form) and the relevant knowledge production rules are being passed to the identified inference/reasoning mechanism. A rule description language that can be used is the Semantic Web Rule Language (SWRL). The output of this process is the produced Knowledge. This step may be bypassed, in case of a request for information aggregation without knowledge production.

Step 7: The produced knowledge or aggregated information may need a post-processing (e.g., filtering). This step is optional.

Step 8: At this stage, the result is being communicated to the ICD function, to find its way to the requesting UMF component. The produced knowledge or aggregated information can be optionally stored in the ISI

function so as to be available for UMF core mechanisms or NEMs when requested/needed. The knowledge production workflow is illustrated in the Figure 27.

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. 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	UMFInformationSpecification, UMFInformation
List of output data	Aggregated Information, represented as UMFInformation
List of non-functional requirements	N/A

Operation 2	**Produce/Build Knowledge**
Description	Produces knowledge out of information, aggregated information or locally-scoped knowledge. Reasoning and inference mechanisms are required for this process.
Constraints	The required information, 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, i.e., UMFInformationSpecification, UMFInformation
List of output data	Knowledge, represented as UMFInformation
List of non-functional requirements	N/A

Trust evaluation (and Certification)

In the traditional network management schemes, the trust of human operators on the behaviour of the network devices was based on two aspects: (i) exhaustive testing and certification activities before deployment at production level, and (ii) the limited capability of the network element for taking decisions and enforcing actions, which make them rather inoffensive for the overall network activity.

The introduction of autonomic elements (NEMs in UniverSelf context) in an operator network implies at the same time the introduction of the uncertainty about the correct functioning of the intelligent devices able to take decisions on their own. Since the decisions taken by NEMs depend on the context, exhaustive certification activities before deployment cannot cover all the possible situations. UMF tries to overcome this with the introduction of trust mechanisms that asses the functioning of the NEM.

Trust evaluation is introduced in the UMF Release 3 as an operation of the Knowledge Production function, since it builds knowledge about the performance of the NEMs. For its proper functioning, Trust evaluation needs as input:

- list of NEMs to be evaluated, and conditions. This information will be received in the form of policies sent by the Governance core block
- trust index threshold: when the trust index drops below the threshold, the operation of the NEM is not considered trustworthy anymore and GOV block should be alerted. This threshold is sent by Governance core block in the form of policies
- NEM static information: parameters on which Trust evaluation for this NEM should be based, which are specified in the NEM Manifest
- information on the NEM decisions and actions (which decision/action, in which context), which requires a registration of this mechanism to the appropriate information from the NEM.
- information on the network state (monitored information before and after the NEM decision)

The information in the three last items can be retrieved through the "Information Collection & Dissemination" function, which allows the access to the NEM Registry and the NEM historical information.

As output, Trust evaluation provides information on the NEMs "trustworthiness", in the form of a trust index. This trust index is a measurement of the performance and reliability of the NEM. When the computed trust index drops below the threshold indicated by the policies, an alert should be sent to GOV through the Call for Governance interface. GOV then can set this untrusted NEM in an "under trial" state, in order to allow the observation of the behaviour and decision making process of the NEM, but without allowing it to enforce actions onto the network. A workflow illustrating this process is shown in Figure 28.

Operation 3	**Evaluate Trust**
Description	Analyzes NEM performance and produces an estimation on the trustworthiness of the NEM, in the form of a trust index.
Constraints	The required information should be available in the storage or can be produced at real-time.
List of input data	List of NEMs to be evaluated, trust index threshold, NEM Manifest, NEM context information, decisions/actions taken by the NEM
List of output data	Instantaneous trust index, overall trust index
List of non-functional requirements	N/A

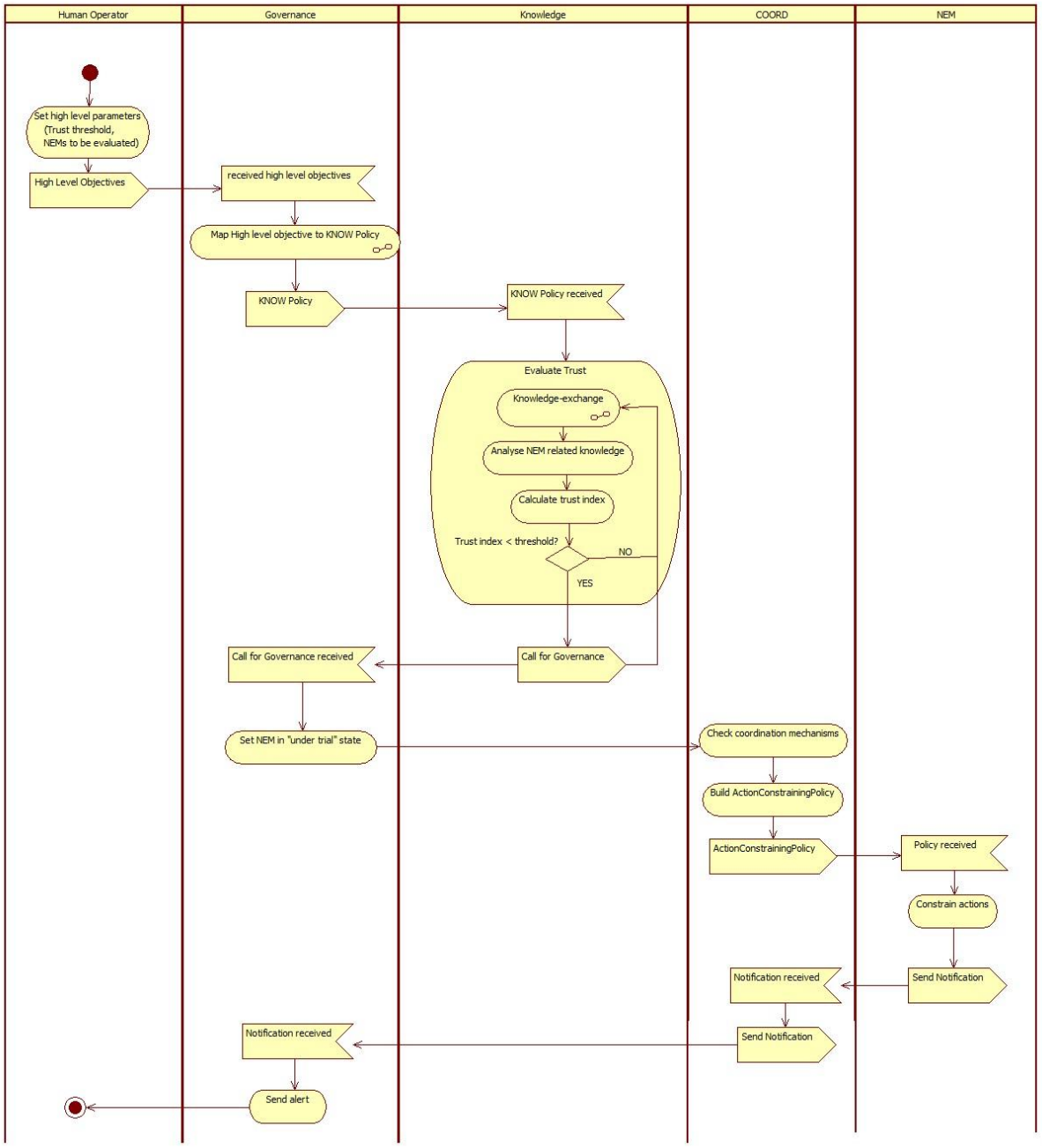


Figure 28: Call for Governance issued by Trust mechanisms

UniverSelf distinguishes between offline and online building-trust procedures. Online means that the operation of a given NEM or set of NEMs deployed at production level are being continuously scrutinized by the trust mechanisms during the operational phase of the NEMs. On the other hand, offline trust refers to an evaluation made prior to the deployment at production level and therefore in conditions that may try to simulate or emulate the context in production. It is worth noting that in both cases the same trust-building mechanisms can be used, with the only difference of the actual context in which the NEM is embedded. The output of those mechanisms is a trust index, qualified as online or offline according to the scenario where the index was calculated.

3.4.6 Information Flow Establishment and Optimisation function

In this section, we discuss the specifications of the information flow negotiation & optimization aspects. These two crucial processes are being overseen from the Information Flow Establishment & Optimization (IFEO) function of the KNOW block. The IFEO function (see Figure 29), besides organizing internal KNOW optimization aspects (e.g., setting filtering or information accuracy objectives), also regulates the information flow based on the current state and the locations of the participating UMF components (e.g., the NEMs producing or requiring information). All relevant communication between the KNOW and the UMF components takes place through the Knowledge Management interface, unless it is otherwise stated.

For clarity purposes, we define the specifications of the IFEO function along with a representative example. We assume the following two NEMs: (a) the Virtual Infrastructure Management (VIM) NEM that provides management & control facilities for virtual infrastructures, including support of traffic monitoring; and (b) the Placement Optimization (PO) NEM that optimizes the data flow over a virtual network through adapting the positioning of communicating nodes (e.g., data servers) in response to the dynamic network conditions. In this example, the VIM NEM provides traffic monitoring information from a particular virtual network to the PO NEM. The PO NEM takes optimization decisions for the network based on this information, i.e., repositions communicating nodes in order to optimize network communication. The information flow negotiation & optimization processes include a number of basic phases, elaborated below:

Phase 1 - Registration: In this phase the NEMs, as part of their registration process with the KNOW block (i.e., described in section 3.6.2), will communicate the following information to the KNOW:

- Information they can offer instantly or after an information collection process.
- Knowledge they can offer instantly or after a knowledge production process.
- Information/knowledge they would require (mandatory or optional).

The above information is embedded in the description of the NEM instance description (see section 3.2.6). All these appear as lists of the UMFInformationSpecification data structure defined in section 3.2.2.*In our example, the VIM NEM registers the information it can offer (e.g., the topology information & measurements on the link loads). This information can be offered instantly (i.e., does not require an information collection process to start, since it monitors the network continuously). The PO NEM registers the same information type as mandatory information required.*

Phase 2 –UMF block requesting knowledge: In this phase a process in a UMF block (like a supervision process in GOV or a KNOW knowledge production mechanism or a COORD Conflict Avoidance Mechanism) demands to register to a given piece of information produced by a given NEM. This information is expressed as a UMFInformationSpecification (optionally enriched with the NEM instance ID, which is then a NEMInformationSpecification). In that case, the Knowledge Management Interface of the requesting entity is calling the KNOW method named RequestInfoRegistration which takes as parameter an object Inheriting from a ManagementInfoSpecification.

Phase 3 - Information Flow Negotiation: In the third phase, the KNOW block through its IFEO function handles a flow negotiation process between entities (i.e., NEMs or core mechanisms) requiring information and those can provide it. The two entities exchange information flow related parameters with the KNOW block, in order to confirm that all information-related requirements can be satisfied under the given constraints. An information flow is either established between the two entities directly or between an entity and the KNOW block itself, in case the requested information is available in the KNOW storage. The negotiation process includes flow-level optimization aspects as well. This phase is composed of the following steps:

Step 1 - Preselecting the information flow ends: Whenever a NEM registers it advertises requested information/knowledge (under the format of UMFInformationSpecification), the KNOW block fetches in its indexing storage the appropriate entity (NEM or Core Mechanism) that can produce the requested information/knowledge. It may either select an entity by considering the type of information/knowledge required or, in case of alternative options, assign the first entity it finds and enlist the other potential choices in a queue. In case the required information is in the KNOW storage, an information flow is created with the KNOW. The same process happens when a UMF entity requests some knowledge, depending on the form of the request (i.e., a fetching from the indexing storage may or may not be required):

- ***NEMInformationSpecification:*** already specifies which is the Instance ID of the NEM producing the information.

- ***UMFInformationSpecification:*** a fetching from the index table is required to pick the appropriate flow ends.
- ***ManagementInfoSpecification:*** then the fetching does not concern finding a flow end, but finding all the flow ends matching the pattern provided by the ManagementInfoSpecification in order to establish as many flows as indexed UMFInfoSpecification objects inheriting from the ManagementInfoSpecification (this corresponds to a supervision mechanism requesting to register to NEM utilities, hence a flow for each NEM capable of advertising its utility will be created). Reversely, KNOW may have postponed flow establishments of some requested information because at the time the request was received, no entity producing this information was registered. In that case, KNOW checks with every received Instance description whether the advertised information matches previously unsolved requests. After that, the IFEO proceeds to Step 2. *In the studied example, the KNOW block preselects the VIM NEM as information source for the PO NEM that acts as the information sink. This selection was based on the matching information URIs referenced in the registered UMFInfoSpecifications data structures from the two NEMs.*

Step 2 -Communicating the negotiation parameters: In step 2, a negotiation process is initiated between the entity requiring information/knowledge (i.e., the information sink entity) and the selected information source entity. The negotiation begins with the two entities communicating additional negotiation parameters to the KNOW block. Specifically, the information sink entity communicates an augmented version of the UMFInformationSpecification with:

- QoS Requirements on the information/knowledge it requires.
- Preferred information communication method (i.e., either push/pull or pub/sub).
- List of Knowledge Exchange interfaces (addresses) on which the information can be received and possibly an internal metric regarding the internal costs to use this information from each of these interfaces.
- REST callback functions that may be required at this end of communication (e.g., in case of an information subscribe method).

In a similar way, the information source entity communicates the following to the KNOW block:

- QoS Constraints on the information/knowledge it can offer.
- Supported(and preferred) information communication method (i.e., either push/pull or pub/sub).
- Whether for this requested information/knowledge an "information collection/knowledge production" process is already activated or needs to be initiated.
- List of Knowledge Exchange interfaces (addresses) on which the information can be provided and possibly an internal metric regarding their internal cost to bring this information up to the interface.
- REST Callback functions for the relevant capabilities (i.e., triggering functions for information collection or knowledge production - if relevant).

Practically, the KNOW block initiates a new negotiation with the execution of the GetSinkNegotiationParameters and GetSourceNegotiationParameters methods of the Knowledge Management Interface. Both methods take as input the specifications of the information to be communicated from the established communication flow, represented as an UMFInfoSpecification data structure. *In the reference scenario, the VIM NEM communicates to the KNOW: (i) the QoS constraints of the topology and link load information it can offer, e.g., monitors information once per 10 secs, and (ii) a number of available Knowledge Exchange interfaces that can provide the information. The PO NEM communicates to the KNOW: (i) the QoS requirements of the required information, e.g., once per 30 secs, and (ii) a number of available Knowledge Exchange interfaces that can receive the information.*

Step 3 - Completing the negotiation: The KNOW block matches information flow requirements with constraints, determines the information flow parameters with flow optimization considerations and then issues a Knowledge Exchange Policy summarizing an information flow contract to both entities. KNOW also stores the Knowledge Exchange Policy through the Information Flow Configuration & Statistics operation of the IFEO function. In case of an unsuccessful negotiation (i.e., the requirements do not match the constraints), it may disengage or trigger a new negotiation:

a) With the same information source entity but lower requirements.

b) With another information source entity that waits in the queue, until the queue is exhausted.

The Information Exchange Policies for the corresponding flow are being produced from the Information Quality Controller operation of the IFEO KNOW block function and include:

- Location/addresses of the participating Knowledge Exchange Interfaces in the information flow.
- Internal KNOW optimization decisions that may impact the information flow (e.g., optimal KNOW aggregation/storage points), in case the KNOW block is the one end of the flow.
- Addresses of triggering callback functions for knowledge production or information collection - if relevant.

These policies are considering the requirements/constraints of the participating entities & the global performance objective coming from the GOV block (i.e., see section3.4.7). The KNOW establishes the information flow using the SetKnowledgeExchangeFlow method of the Knowledge Management Interface, that takes as an input the decided Information Flow Exchange Policies, represented as a KnowledgeExchangeFlowSpecification data structure.

The decided Information Exchange Policies are being applied to the network through the respective NEMs or communicated to the KNOW functions they are associated with. Since the appropriate context environment for the new information flow is prepared, a suitable path between the participating nodes is established next. This process considers the locations of the entities producing and requiring information and the required KNOW nodes (e.g., aggregation points, storage points etc) as well as the potential traffic characteristics. After that, the Knowledge Exchange interface can be accessed anytime from the information sink entity to receive the needed information/knowledge. *In our reference scenario, the KNOW block matches the information flow constraints (e.g., supported monitoring rate) of the VIM NEM with the information flow requirements from the PO NEM. Then it selects the most appropriate Knowledge Exchange interfaces to communicate the information from the VIM to the PO NEM. A new information flow contract is established and communicated to the two NEMs and stored in the KNOW block. The information flow is established and the PO NEM can retrieve the required information from the VIM NEM via the appropriate Knowledge Exchange interface. The PO NEM can now begin taking network optimization decisions using that information.*

KNOW-level Optimization: Furthermore, KNOW supports a global optimization process that is triggered periodically or when a global performance objective change is requested from the GOV. This process takes optimization decisions using the aggregated information from the configuration of all established information flows and is related with a restructuring of the KNOW functions themselves. In other words, global-optimization algorithms may discard or update Knowledge Exchange Policies enforced for established information flows. It takes as an input the global picture of all the established information flow contacts and provides as an output different contracts aligned better to the new updated demands (e.g., a new received global objective). This process may initiate re-negotiations that include requesting again from the entities what their requirements and capabilities are. For example, the distributed KNOW nodes may be increased, decreased or repositioned in order to accommodate all established information flows and the global optimization goal better.

An example optimization process that is triggered from the IFEO function and regulates the information flow based on the current state and the locations of the participating UMF components is shown in Figure 29. In particular, the IFEO controls information collection handled from the ICD function, information aggregation, and aggregation node placement. Furthermore, it guides a filtering system for information collection and aggregation points that can significantly reduce the communication overhead.

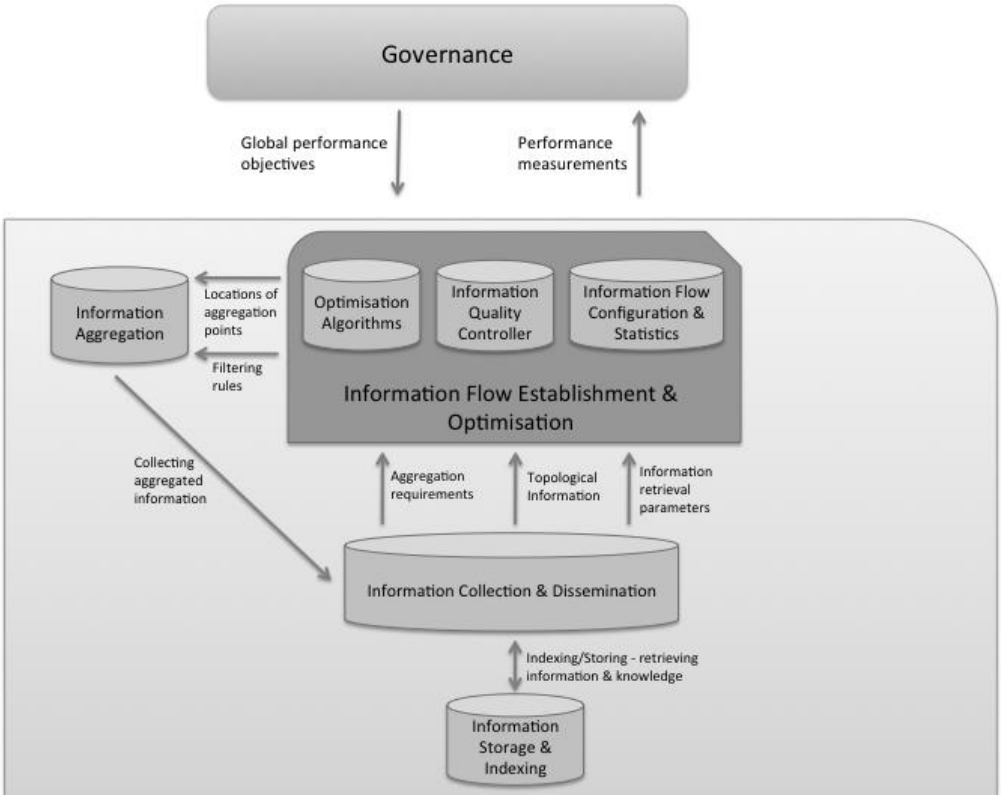


Figure 29. Overview of the Information Flow Establishment and Optimization Function.

Lifecycle and management of direct NEM2NEM flows

This paragraph discusses issues related to the direct information flows between NEMs (i.e., NEM2NEM) against those having KNOW in the middle (i.e., NEM2KNOW). Additionally, it specifies the lifecycle and management of NEM2NEM flows. Main concerns on the NEM2NEM flow scheme are: "*When and why would that be chosen over the NEM2KNOW equivalent?*", "*When and how would a NEM2NEM flow be stopped, i.e. switched to NEM2KNOW?*".

Why is NEM2NEM advantageous over NEM2KNOW?

- Offloads the network: NEM2KNOW uses at least two additional hops.
- Offloads the KNOW block storage & processing capabilities: the KNOW block has to temporarily store-and-forward information received from the source to the sink and in some cases to provide longer-term storage, validation and/or additional processing of data.
- Improved delay/response time (for delay-sensitive data/information): imagine data/info with time-to-live less than 100ms or 50ms, then two extra hops impact the on-time delivery.
- Confidentiality and privacy purposes: information exchanged might be confidential to the source and sink NEMs, so they have to establish a private, secure communication channel.
- It allows NEMs to use communication protocols other than the ones provided/supported by the UMF to better suit their needs, e.g., a design choice for real-time data communication might be to use RTP or another application-level protocol that builds on top of UDP.

Managing the NEM2NEM flow

After a NEM2NEM information flow has been established, the KNOW block should be continuously auditing it. One reason is the potential optimization opportunities that might appear in case additional consumers of the same knowledge join. Another reason is trust building; the KNOW block monitors the knowledge exchange at an abstract level and infers (e.g. by receiving feedback from the consumer NEMs) on the quality and responsiveness of the communication as well as other trust-related performance metrics.

So, the KNOW block monitors every flow that is not routed through the block itself. Notification messages (i.e. events) on the type of information exchanged, the id, the lifecycle as well as the status of the flow are being

delivered to the KNOW. The scheme includes bi-directional triggering of such events, i.e., the Flow Manager operation of the KNOW IFEO function explicitly requests an update on the status of a particular flow, while the Information Sink triggers the event of termination of a flow and thus undertakes to inform the flow manager. Although such messages are sufficient for the flow manager to infer whether a source has any sinks attached to it, there could be an explicit notification by the source for that purpose.

Table 35: NEM2NEM Information Flow Management Messages

Message Name	Sender	Receiver	Description
FlowEstablishmentNotification	Information Sink	Flow Manager	To inform the flow manager for the successful establishment of the NEM2NEM flow.
FlowKeepAlive	Information Sink	Flow Manager	Regularly sent by the information sink to inform the flow manager that a previously established flow is still alive.
InfoSourceEvaluationReport	Information Sink	Flow Manager	The flow source performance (quality, responsiveness) metrics are being informed to the Flow Manager by the source. This evaluation is performed by the Information Sink and is subject to the trust index of the sink itself.
InfoSourceStatusNotification	Information Source	Flow Manager	To inform the flow manager for the information source status. For instance, in case the source is running out of resources, it uses this message to alert the Flow Manager to handle the situation, e.g., to terminate the NEM2NEM flow and establish a NEM2KNOW equivalent.
InfoSourceStatusNotificationRequest	Flow Manager	Information Source	The Flow Manager requests the generation of an InfoSourceStatusNotification message from the Information Source.
FlowReroute	Flow Manager	Information Source / Sink	The Flow Manager instructs the NEMs to re-route the flow through a particular KNOW node.
FlowRerouteAck	Information Source	Flow Manager	Upon the reception of a FlowReroute message, the Information Source waits until a time interval expires. Within that time-period, sinks are expected to stop requesting info. In case they do not, the source will forcefully interrupt all active flows upon expiration of the time period, and then send this message back to the Flow Manager.

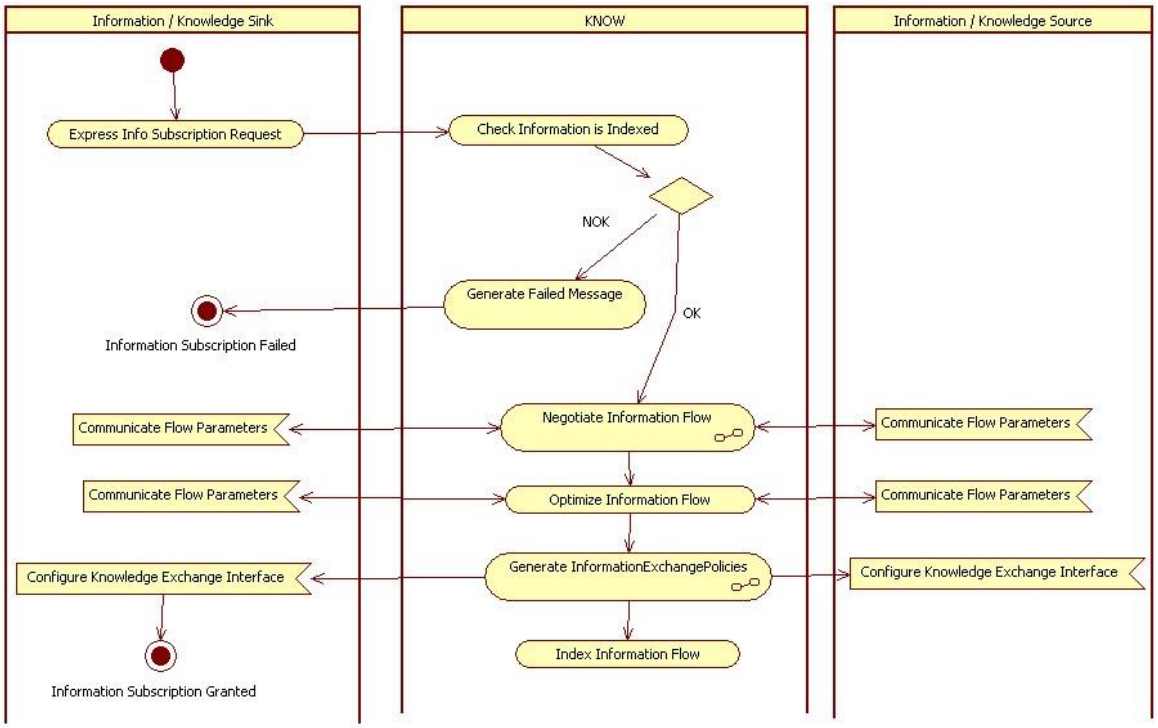


Figure 30. Information subscription workflow diagram

List of "Information Flow Establishment and Optimization" operations:

Information flow negotiation, Information flow optimization, Information quality control, manage information flow

Operation 1	**NegotiateInformation flow**
Description	Responsible for the establishment of every information flow. This process takes place proactively during a NEM registration or configuration update. The flow is optimized from the information flow optimization operation.It can be triggered directly, in case of core blocks requesting exchange of knowledge.
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	List<UMFInformationSpecification>
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	List<UMFInformationSpecification objects>
List of output data	InformationExchangePolicies
List of non-functional requirements	N/A

Operation 3	**Information Quality Control**
Description	Responsible for information flow optimization decisions, including flow-level and global-level optimizations.
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	List<UMFInformationSpecification>, information flow constraints, information flow requirements, locations of interface attachments, InformationExchangePolicies of existing flows.
List of output data	local KNOW optimization decisions, InformationExchangePolicies or List<InformationExchangePolicies>, in case of global-level optimizations.
List of non-functional requirements	N/A

Operation 4	**ManageInformation Flow**
Description	The IFEO function audits the established information flows through the Information Flow Management operation. This validates that the flow behaves in the way it is described in the information flow contract.
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	FlowEstablishmentNotification, FlowKeepAlive, InfoSourceEvaluationReport, InfoSourceStatusNotification, FlowRerouteAck
List of output data	InfoSourceStatusNotificationRequest, FlowReroute
List of non-functional requirements	N/A

3.4.7 Governing Knowledge Exchange

The Governance block performs high-level management of the KNOW block through setting a global performance optimization goal using the Knowledge Management Interface (as part of the Information Quality Control operation). The KNOW block takes and applies local optimization decisions (transparently to GOV) for a number of information/knowledge manipulation aspects, in order to realize the specified goal. The GOV block uses the Knowledge Exchange Interface to collect performance measurements for the KNOW block functions' operation in the same way with every other UMF component; a process that monitors the KNOW compliance to the specified goal.

In the first two columns of Table 36: Example Optimization Goals, Related Local Optimization Decisions and Associated Performance Metrics, we give examples of optimization goals and local optimization decisions being taken within the KNOW block to meet the specific goal. In the third column, we give examples of relevant performance measurements that GOV can monitor, in order to check the compliance of KNOW functions performance to the particular goal. This may mean an implicit policy setting to NEMs as well. For example, a request from a NEM to retrieve a fresh value from another NEM may be denied and a cached value may be fetched instead. In general, the optimisation goal coming from GOV may override the requirements specified from the NEMs regarding information/knowledge handling.

Table 36: Example Optimization Goals, Related Local Optimization Decisions and Associated Performance Metrics

Optim. Goal	**Local Optimization Decisions**	**Performance Measurements**
Reduce Response Time	- Place extra KNOW nodes supporting the ICD function close to the entities communicating information - Use KNOW nodes that support many or all of the KNOW	Response Time, Communication Overhead, Information Accuracy,

	functions (i.e., to avoid distribution of information to remote KNOW nodes). - Use storage technologies that support information caching. - Use responsive knowledge production mechanisms. - Apply relaxed information accuracy objectives in order to minimize communication cost. - Use information dissemination mechanisms that minimize response time (e.g., prefer push to pull). - Prefer direct NEM2NEM communication to communication through KNOW.	Knowledge Production related indices.
Reduce Processing Cost	- Use less KNOW nodes - Use lightweight knowledge production mechanisms. - Apply relaxed information accuracy objectives in order to minimize the need for processing.	Processing Cost, Knowledge Production related indices, Information Accuracy.
Reduce Communication Overhead	- Place extra KNOW nodes supporting the ICD function close to the entities communicating information - Use KNOW nodes that support many or all of the KNOW functions (i.e., to avoid distribution of information to remote KNOW nodes). - Use storage technologies that support information caching. - Apply relaxed information accuracy objectives in order to minimize communication cost. - Prefer direct NEM2NEM communication to communication through KNOW.	Communication Overhead, Information Accuracy.
Improve Information Accuracy	- Use accurate knowledge production mechanisms. - Apply stringent information accuracy objectives. - Prefer communication through KNOW, in order to exploit easier all available information. - Disable storage caching.	Information Accuracy, Knowledge Production related indices.
Improve Energy Efficiency	- Place the optimum KNOW nodes supporting the ICD function close to the entities communicating information - Use KNOW nodes that support many or all of the KNOW functions (i.e., to avoid distribution of information to remote KNOW nodes). - Use energy efficient storage technologies that support information caching. - Use responsive and lightweight knowledge production mechanisms. - Apply relaxed information accuracy objectives in order to minimize communication cost. - Use energy efficient information collection / dissemination mechanisms. - Prefer direct NEM2NEM communication to communication through KNOW.	Response Time, Communication Overhead, Processing Cost, Energy Consumption, Information Accuracy, Knowledge Production related indices.

In Table 37. KNOW Optimization Information Model Representation shows the set of parameters that GOV can use to define optimization goal policy towards KNOW.

Table 37. KNOW Optimization Information Model Representation

	Name	Description
Policy	KNOW Optimization Goal	The KNOW optimization goal, specified from the Governance block.
Policy Parameters	optGoalId: Number	The Id of the particular performance optimization goal.
	optGoalName: String	The name of the optimization goal.
	optGoalParameters: Depends on the goal	The parameters of the optimization goal
	optGoalLevelofEnforcement: {Low, Medium, High}	The level of goal enforcement. For example, high level defines a critical situation where the enforcement should override all other relevant decisions been taken.

3.5 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.

To this end, COORD can define these conditions/constraints and influence the behaviour of NEMs at certain time-scales; that are:

- During registration of NEMs, when a NEM is firstly introduced into the system
- During "specific behaviour requests" from GOV, while a NEM is already in operational mode and GOV needs to alter its behaviour or even its working mode; in such cases COORD may need to update the behaviour of this NEM and also other NEMs that may be affected by such alterations.
- During runtime, while a NEM is in operational mode and needs guidance due to the nature of the specific coordination mechanism that manages and controls its behaviour (e.g. as it will be shown later, certain mechanisms for managing NEMs during runtime require from NEMs to actually suspend their operation and simply enforce specific network parameters when requested; in such cases NEMs require continuous guidance during runtime to retrieve and enforce these specific network parameter values)
- During runtime, when a problematic situation is identified. In such cases, COORD can "revisit" its settings and produce alternative configurations in an attempt to remedy these problematic situations.

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

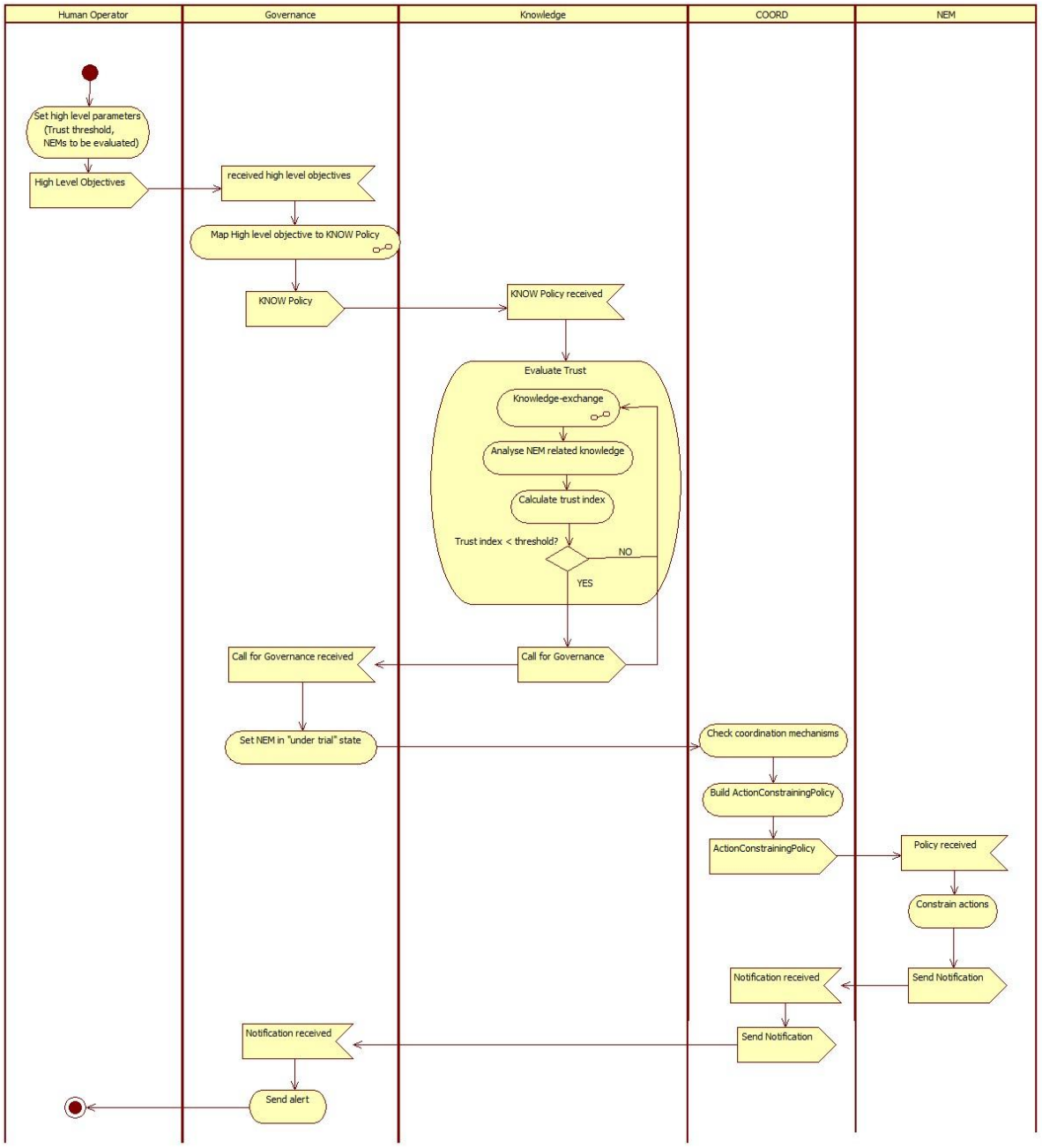


Figure 28: Call for Governance issued by Trust mechanisms

UniverSelf distinguishes between offline and online building-trust procedures. Online means that the operation of a given NEM or set of NEMs deployed at production level are being continuously scrutinized by the trust mechanisms during the operational phase of the NEMs. On the other hand, offline trust refers to an evaluation made prior to the deployment at production level and therefore in conditions that may try to simulate or emulate the context in production. It is worth noting that in both cases the same trust-building mechanisms can be used, with the only difference of the actual context in which the NEM is embedded. The output of those mechanisms is a trust index, qualified as online or offline according to the scenario where the index was calculated.

Information Flow Establishment and Optimisation function of KNOW

List of Coordination block functions

- Orchestration
- Conflict identification
- Optimization and Conflict Avoidance

3.5.1 Information model of COORD

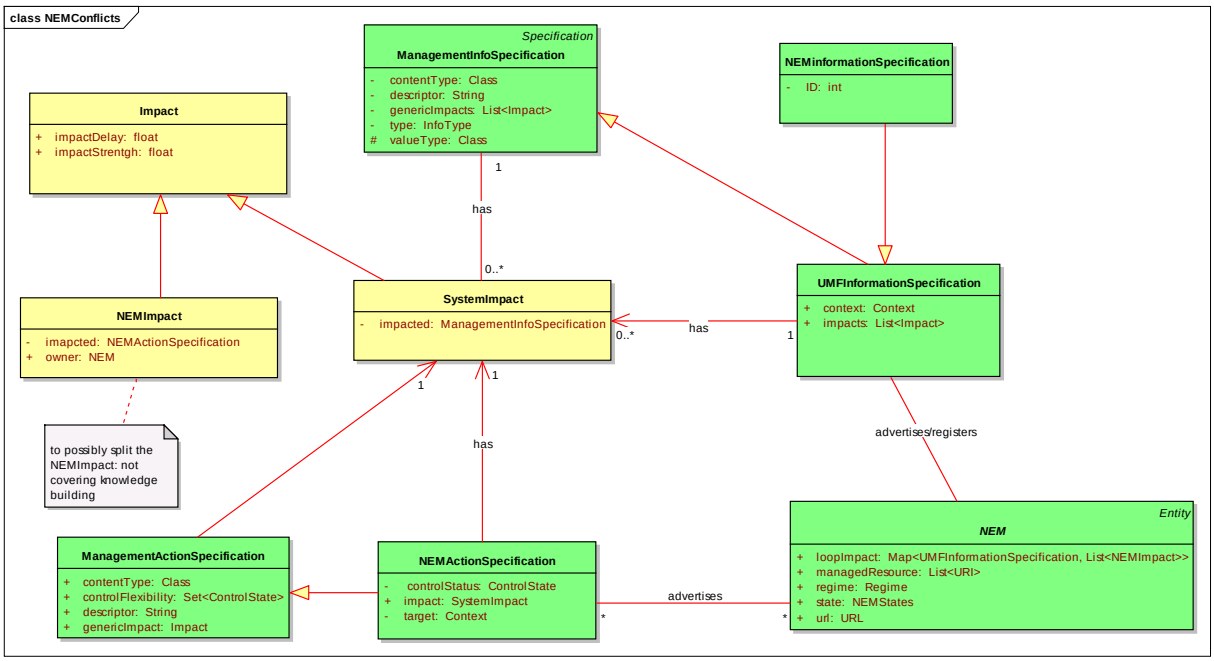


Figure 31. Information model of impacts

The information model of impacts describes how the conflict identification mechanisms modelizes that some actions are impacting metrics and that the metrics are themselves impacting other metrics through a cascade effect.

An example of this is: The ManagementActionSpecification - modifying the pilot power of a LTE antenna - is impacting the ManagementInformationSpecification – pilot power of a LTE antenna – which is itself impacting the ManagementInformationSpecification – coverage of a LTE antenna – which is itself impacting the ManagementInformationSpecification – users load of a LTE antenna.

The information model of conflicts describes how the conflict identification mechanism modelizes the conflicts between two (or more) NEM instances.

There is a distinction between the static conflicts which are detected at registration time of NEMs and this is being based on the NEM instance description (see section 0), and the dynamic conflicts which are identified at runtime, and this is based on inferring dependencies between metrics.

The static conflicts are identified when at least two different NEM instances have overlapping cycles[10], hence a conflict is modelized as a list of overlapping cycles, and the overlapping points in the cycle are identified.

[10] Cycles correspond to the graphical representation of control loop, where half of the loop is represented by NEM inputs (UMFInformationSpecification) towards NEM outputs (NEMActionsSpecifications), and where the second half of the loop is represented by cascading impacts.

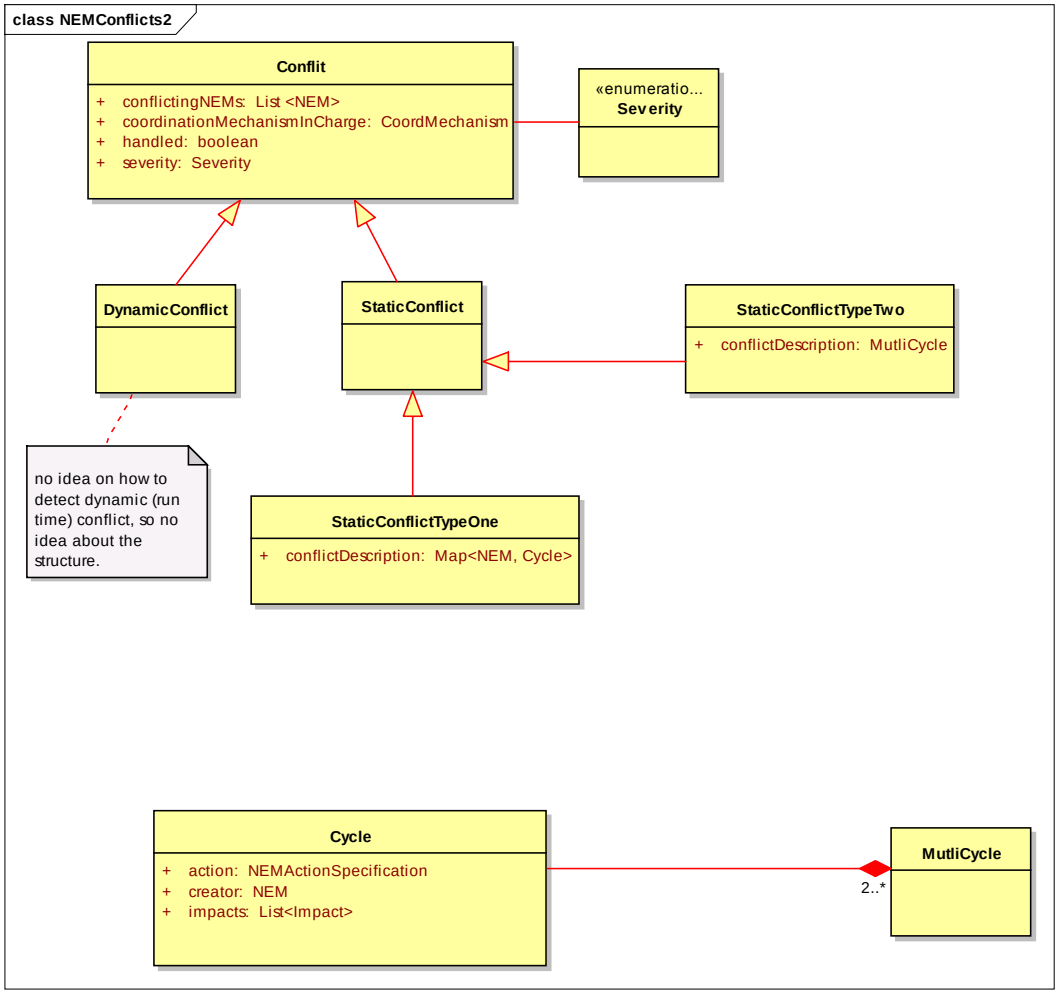


Figure 32. Information model of conflicts

3.5.2 Machine readable description of COORD

The description of COORD should list the configuration options of COORD so that GOV can affect COORD's behaviour. So far three distinct configuration options have been identified.

The first one is related to what COORD will consider as a potential conflict situation; as it will be shown in a later section, parameter value, metric value and loop types of conflicts can be considered.

The second configuration option relates to the "sensitivity" of COORD when it identifies potential conflicting situations. This configuration option can affect several internal COORD mechanisms and operations in different ways.

The third configuration option relates to the orientation of COORD on a per network segment basis; since certain COORD mechanisms require weighting/prioritizing of NEMs, this configuration option can be used to guide this weighting/prioritizations -when needed- based on the operator's overall goals.

The above are reflected in the following example.

```
<?xml version="1.0"?>
<eu.univerself.nem.Manifest>
    <ConfigurationOptions>
        <SpecificCOORDPolicy>
            <name>Conflict Type to be Addressed</name>
            <PossibleValues>{Parameter, Metric, Loop, All}</PossibleValues>
            <defaultValue>All</defaultValue>
        </SpecificCOORDPolicy>
            <SpecificCOORDPolicy>
                <name>Sensitivity</name>
                <PossibleValues>Numeric [0, 1]</PossibleValues>
                <defaultValue>0.5</defaultValue>
        </SpecificCOORDPolicy>
        </SpecificCOORDPolicy>
            <SpecificCOORDPolicy>
                <name>Orientation</name>
                <PossibleValues>In conjunction with what the NEMs declare under Orientation</PossibleValues>
                <defaultValue>(LTE, energy efficiency), (IP/MPLS, load balancing)</defaultValue>
        </SpecificCOORDPolicy>
    </ConfigurationOptions>
</eu.univerself.nem.Manifest>
```

3.5.3 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
List of output data	
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	
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	

3.5.4 Conflict identification function

The role of conflict identification is to -as the term implies- identify potential conflicting situations that may arise due to the concurrent operation of NEMs and, as such, drive the way that COORD handles NEMs in terms of altering their default behaviour and/or constraining their actions.

In the context of the project the following three types of situations are regarded as giving rise to potential conflicts:

- parameter value conflict,
- metric value conflict, and
- loops

Parameter value conflict is the easier to detect; it refers to the situation where multiple NEMs have control over the same exactly network parameter (e.g. antenna pilot power, link weight setting in OSPF, etc.) As a result there exists the possibility that if such NEMs are left unattended, it may simply lead to the situation where each one of them will be setting a specific value to a network parameter, only to have other NEMs override this setting with their own preferences. In cases this happens it will mean that none of the NEMs will be performing as intended and that there is a high chance of oscillatory behaviours with the network parameter of interest oscillating along a wide range of values.

Metric value conflict refers to the situation where a metric used as input from one NEM is affected by modifications by other NEMs. This can constitute a problem for example if there is a NEM A that uses metric Z (e.g. utilization) as an input and it is using a reinforcement-like type of learning and optimization. This means that NEM A inherently makes the assumption that the actions itself takes are responsible for the evolution of utilization and relies on the observations of utilization in order to learn and adjust its behaviour. If however there is a NEM B that takes actions affecting the utilization, this will mean that the assumptions made by NEM A were wrong. As a result, NEM A will learn and possibly converge to a wrong behaviour; even worse, if NEM B starts behaving differently or is removed by the system altogether, then NEM A will have to again (from scratch in case of reinforcement learning) start learning a new behaviour. Similar problems can happen in general whenever the changes made by NEM B try to "drift" NEM A away from its nominal standalone behaviour and/or invalidate any assumption that NEM A makes about its inputs.

In addition a potential problematic situation may arise when NEM A can cope with any changes to its inputs by NEM B, but the output of NEM A "feeds" NEM B forming a loop; either explicitly/directly or implicitly through a cascade of other NEMs leading eventually back to NEM B. This may lead to oscillations with NEMs -depending on their behaviour- resorting to frequent and "in the opposite direction" actions between successive invocations.

However metric value conflicts and loops are not straightforward to detect for two reasons. First, there might be a cascaded set of actions by a set of NEMs which actually leads to an input metric of another NEM being affected, as mentioned above. For example it may be a NEM C which affects NEM B which, in turn, affects NEM A, which may also feed back to NEM C. This means that it would not be enough to consider NEM A and NEM B alone as conflicting, but NEM C should be also taken into account. On the other hand though such an approach

would lead eventually to considering that "everyone somehow affects everyone else" so there needs to be a "stopping point" in this process. The second reason is that there exist cases where an input of NEM A being affected by the actions of NEM B do not necessarily constitute a conflict but a normal situation where NEM A is expected to react to changes in that input. For example, a traffic engineering NEM expects that it will have either periodically or based on some other triggering event (link overload) to redistribute traffic along the available paths of a core network domain. Changes to the link load by another NEM (e.g. a NEM running at the access network and affecting the way the traffic enters/leaves the core domain) are not to be treated as a conflict but rather as a normal situation.

The first way to intercept potential conflict situations is during design time, where design time refers to the time a NEM is firstly "inserted" to a NEM deployment scenario. This type of conflict identification will be referred to as "static conflict identification" meaning that its role is to identify potential conflict situations before they can actually occur, based on the NEMs behavioural specification. The second way to intercept conflict situations is during runtime through the use of monitoring procedures. This latter approach, which will be referred to as "dynamic conflict identification" is mean to complement static conflict identification and "catch" conflicting situations that were not picked up at design time.

More details and examples of both static and dynamic conflict identification mechanisms are given in Section 3.3.

The operations of the conflict identification function drive the behaviour as well as the triggering itself -for the dynamic conflict identification case- of the "Identify NEM Coordination Needs" operation which belongs to the Optimization and Conflict Avoidance function.

3.5.5 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 Trigger events and KPI/NEM parameter information from the Dynamic Conflict Identification mechanism Trigger events from "Request specific behaviour for NEMs" requests from GOV.
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
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	

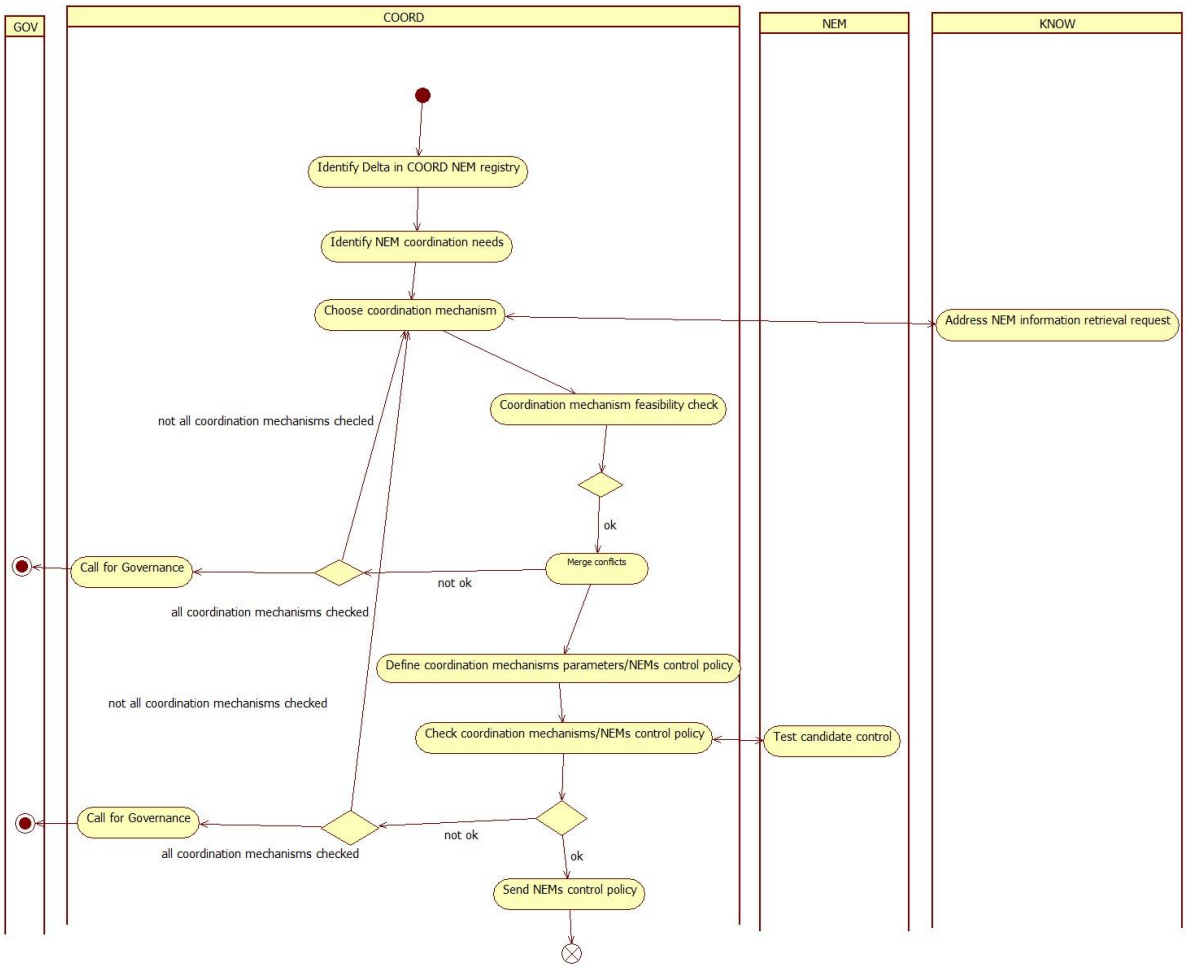


Figure 33. Manage conflicts activity diagram (NEM registration triggered)

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 Figure 10. Information model of Policies regarding NEMs.).

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.

Below is provided the workflow detailing how an entity sends a policy to the NEM, and what is the expected behaviour of the NEM when receiving the policy.

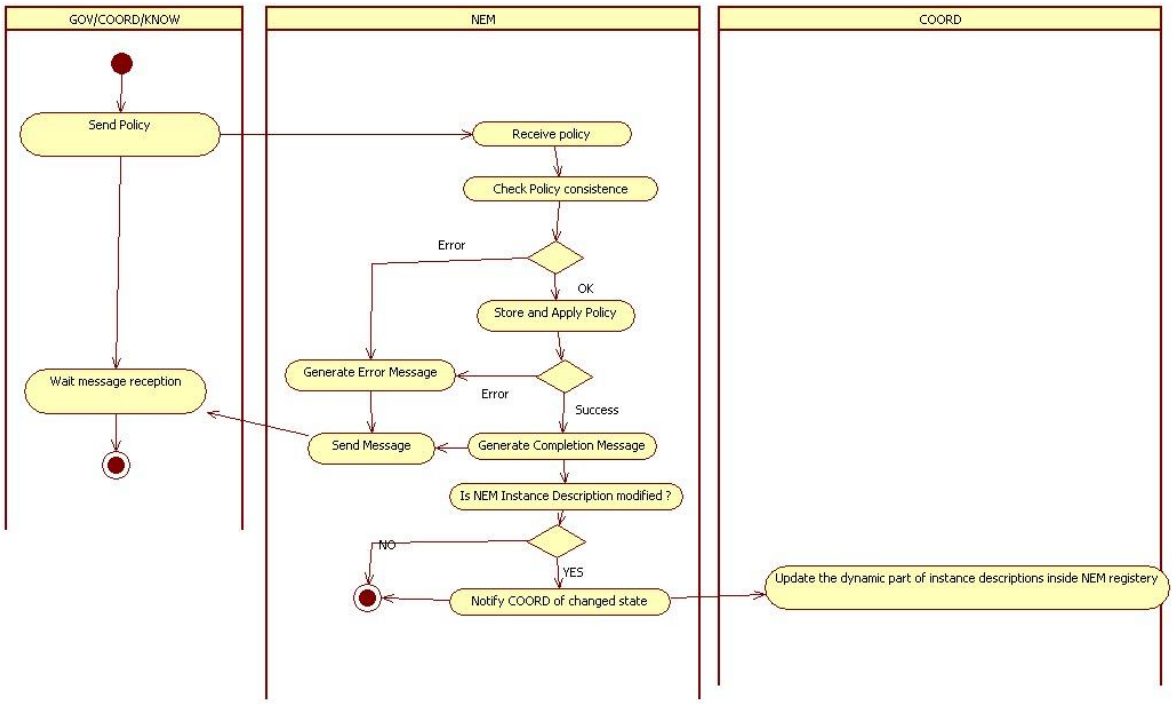


Figure 34. Set policy activity diagram

3.6 Interfaces

3.6.1 Summary of 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 Get NEM Instance Description Revoke NEM Mandate Get Manifest

Interface Name	**GOV -COORD interface**
Description	This interface encapsulates three different operations: • The high level objectives of human operator are also transformed to a set of low-level policies for the coordination of the NEMs. These policies, namely Specific Coord Policies, should be enforced onto COORD core block. • In those cases when Governance needs to set a NEM in a specific state ('under trial' or 'normal operation'), it requests COORD to modify the specific behaviour of a given NEM • 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)	SendSpecific Coord Policies Request Specific Behaviour of a NEM 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; high level objectives, including coordination and knowledge policies; the supervision of the status of network,

	NEMs, and other UMF core blocks
List of operations (exposed operations)	Define Service Define High Level Objectives Supervise Network Status Send Notification

Interface Name	**GOV -KNOW interface**
Description	Interfaces offered between Knowledge and Governance. Governance uses it on one hand to set optimization goals and trust policies to KNOW block, and in the other hand to request information from Knowledge that is needed by the supervisionoperation. This information can be both network/context related but also NEM related
List of operations (exposed operations)	Request NEM information Request context/network information Send KNOW policies Call for Governance

3.6.2 Focus on the Knowledge Exchange and Knowledge Management Interfaces

The ***Knowledge Exchange Interface*** offers basic information/knowledge manipulation functionalities to the UMF entities. This allows a unified information/knowledge exchange process in all types of participating entities. The Knowledge Block uses a separate interface to communicate with other core services and NEMs for aspects related to the management of knowledge exchange functions and established information flows. For example, a policy for a global optimization goal can be set from the Governance block, which triggers a number of configuration changes in the knowledge functions. This interface is called ***Knowledge Management Interface*** and is associated with both internal KNOW block management activities and external to the KNOW information flow configurations.

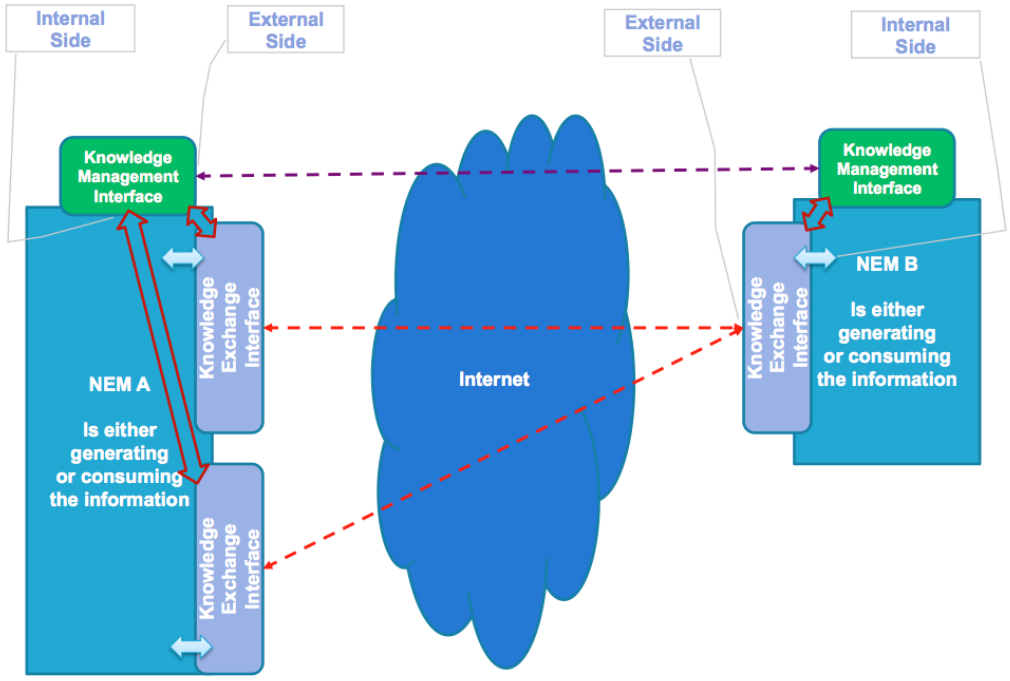


Figure 35. The Knowledge Exchange and Knowledge Management Interfaces

In Figure 36, we show how the two knowledge interfaces participate in the establishment of the information flow between a NEM that produces information (NEM A) and a NEM that consumes it (NEM B). The two NEMs negotiate information flow related parameters with the KNOW block through the ***Knowledge Management interface*** and its corresponding attachment points. The Knowledge block receives the parameters and takes configuration decisions for the new flow. After that, the NEM A can start communicating information to NEM B using the ***Knowledge Exchange interface*** and its associated attachment points.

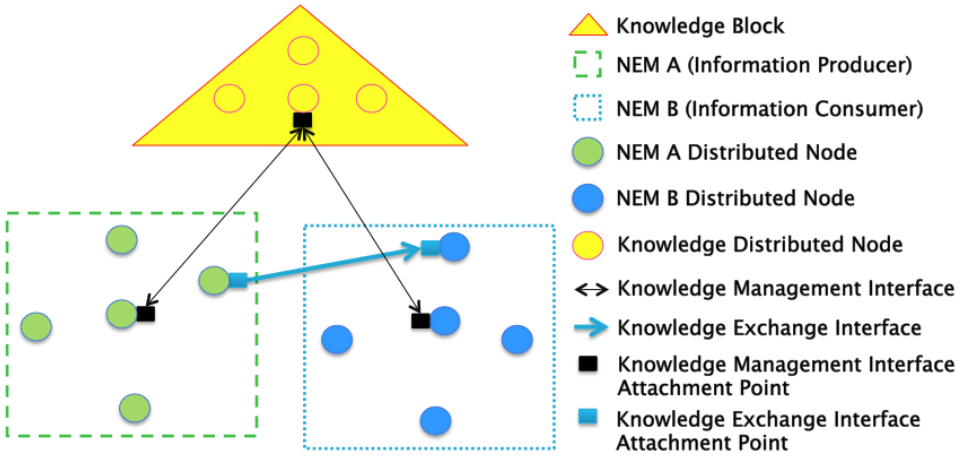


Figure 36. The Role of Knowledge Exchange and Knowledge Management Interfaces in the Information Exchange

The communication of information through the ***Knowledge Exchange interface*** is based on either pull or pub-sub method. Figure 37 illustrates the knowledge exchange process using the pull method. The pub-sub method is illustrated in Figure 38 (i.e., the publish part). The information subscription diagram illustrated in Figure 30 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 have actors that are named knowledge sink and knowledge source. This means these workflows are characterizing the behaviour (Figure 37 and Figure 38) and the configuration (Figure 30) of a Knowledge Exchange Interface, 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. 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.

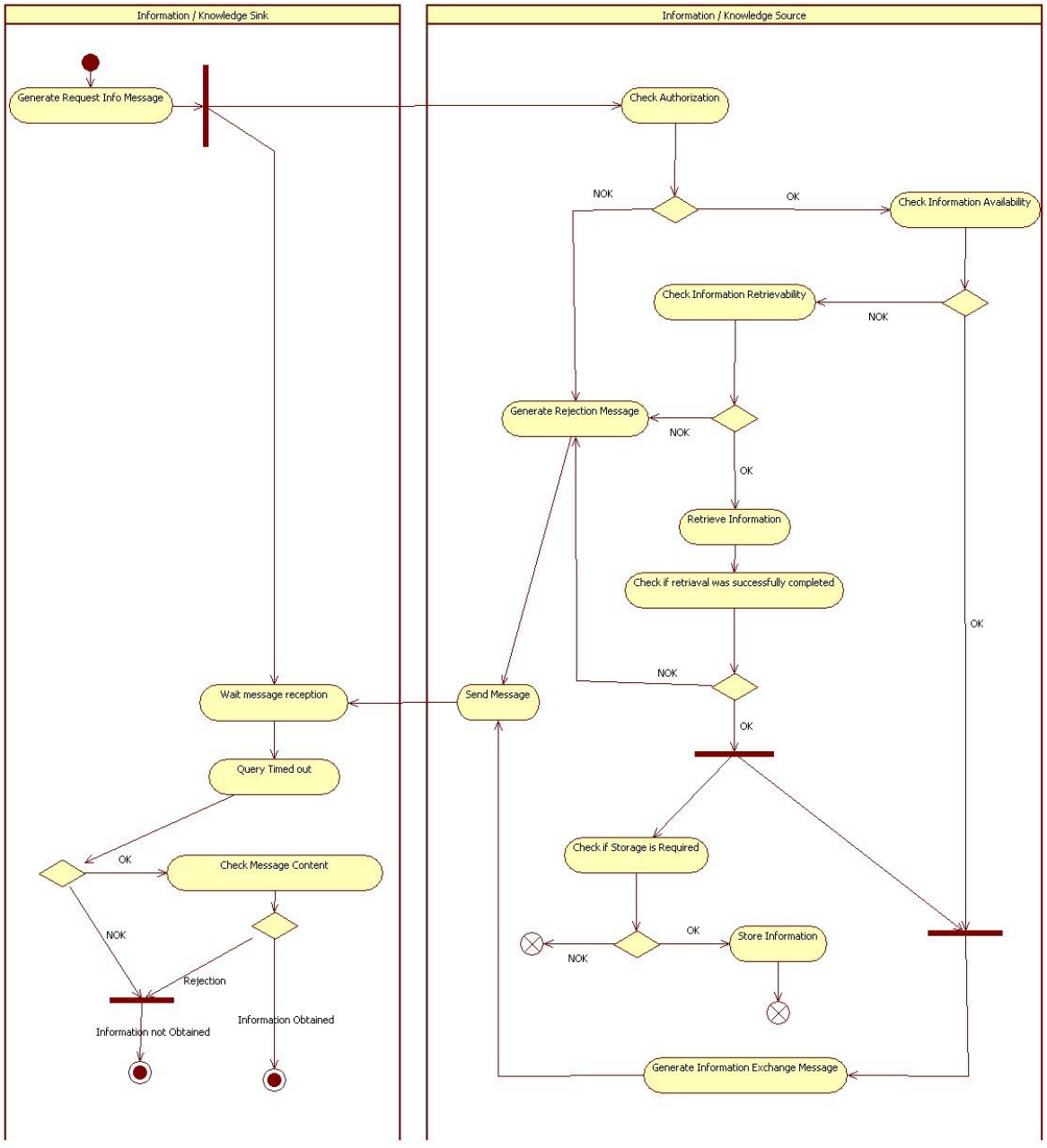


Figure 37. Knowledge Exchange workflow diagram using the Pull method.

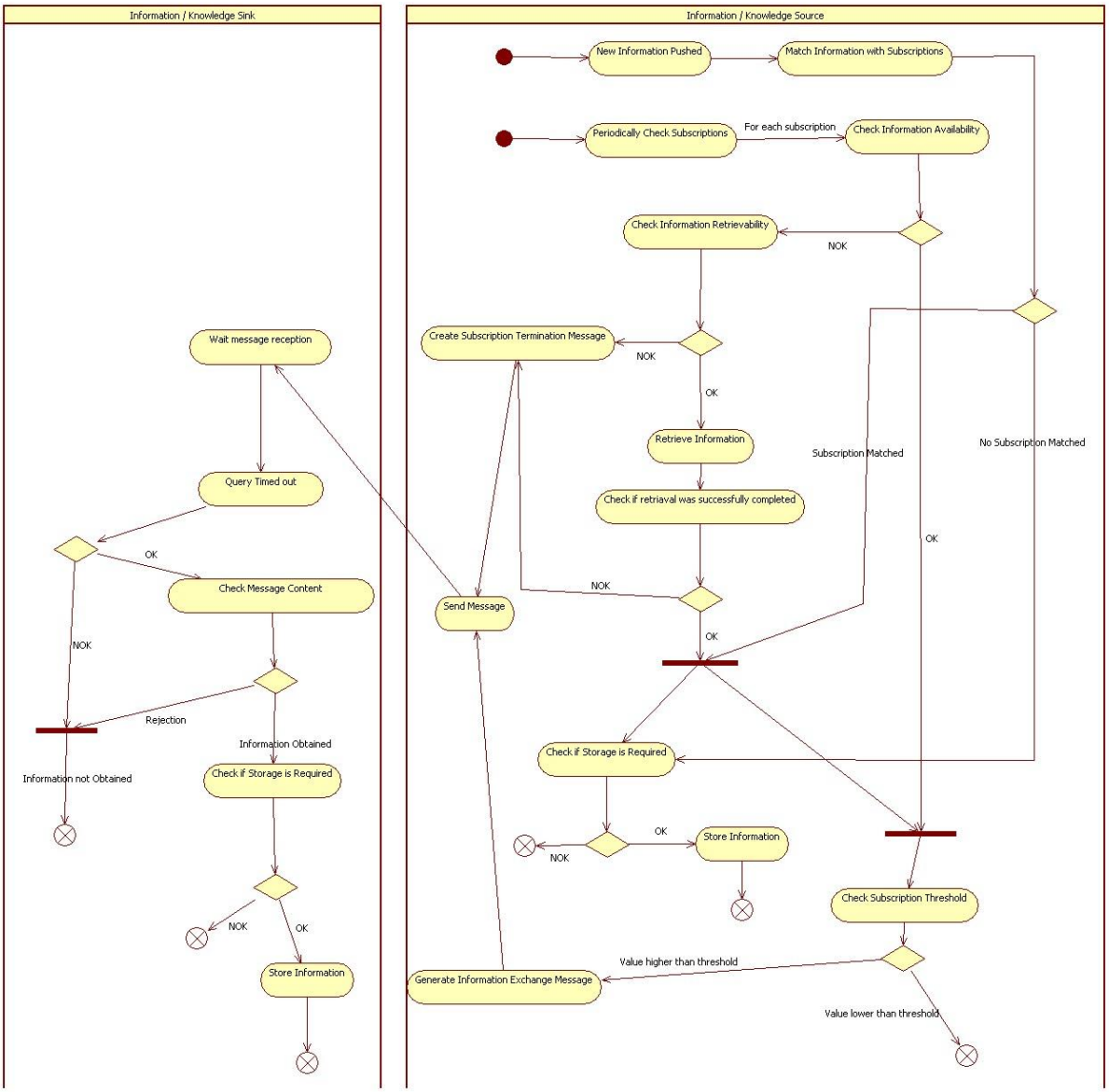


Figure 38. Knowledge Exchange workflow diagram using the Pub-sub method.

A brief description of the two Knowledge interfaces can be found in the following two tables.

Interface Name	**Knowledge Exchange Interface**
Description	This interface is part of KNOW block and is responsible for exposing KNOW operations to the other UMF components, which are related to information exchange
List of operations (exposed operations)	Information collection Information sharing Information retrieval Information dissemination Information aggregation Knowledge production

Interface Name	**Knowledge Management Interface**
Description	This interface exposes to the other UMF core blocks and the NEMs those KNOW operations that are related to management issues of the KNOW and the information flows it handles. For example, the information flow negotiation, the NEM registration to the KNOW and the specification of a global performance goal from the GOV.
List of operations (exposed operations)	Information flow negotiation Information flow optimization Information flow management Information quality control NEM registration to the KNOW

4. UMF core mechanisms

4.1 Governance mechanisms/tools

4.1.1 Translation mechanisms

Policy translation is considered as one of the main operations performed in the Policy Derivation and Management (PDM) function. Its purpose is to translate business policies derived from the HLOs defined by the operator into NEM policies. Towards this direction, the procedures of detection and specification of the kind of information that is conveyed by the policy of the respective policy level is triggered. Also, the transformation of this information in the defined form for the targeted policy level based on the context information takes place. However, the definition of coherent information that is exchanged between the different entities across the different layers, administrative domains and network segments should be considered.

Specifically, the proposed policy translation methodology is based on a three-fold approach in order to maximize automation, retain low complexity and high preciseness, while being highly reusable. Firstly, the operator's High Level Objectives are classified into categories (e.g. QoE, Energy Efficiency), while a set of KPIs are assigned to each category. Secondly, the translation process is not realised from scratch (as in the vast majority of translation studies). In the proposed approach the translation is led by a set of guidelines which are described in a set of policy templates stored in a policy template pool. These templates have the form of xml document with predefined xml schema. An indicative example of such a policy template is also presented below.

```
<?xmlversion="1.0"encoding="UTF-8"?>
<PolicySets>
    <PolicyRulepolicyName=""policyLevel="">
        <PolicyEvents>
            <PolicyEventAtomiceventType=""/>
        </PolicyEvents>
        <PolicyConditions>
            <PolicyConditionAtomic>
                <PolicyStatement>
                    <PolicyVariable></PolicyVariable>
                    <PolicyOperatoropType=""/>
                    <PolicyValue></PolicyValue>
                </PolicyStatement>
            </PolicyConditionAtomic>
        </PolicyConditions>
        <PolicyActions>
            <PolicyActionAtomic>
                    <PolicyStatement>
                        <PolicyVariable></PolicyVariable>
                        <PolicyOperatoropType=""/>
                        <PolicyValue></PolicyValue>
                    </PolicyStatement>
            </PolicyActionAtomic>
        </PolicyActions>
    </PolicyRule>
</PolicySets>
```

In the considered example, a set of high level objectives (i.e. User Class, Availability) comprise the values of the predefined "Policy Variable" and "Policy Value" elements. Thirdly, the modelling of high level objectives and translation process is realised by the use of ontologies (OWL) and ontological rules (SWRL). Ontologies constitute a means to capture information and organise information and knowledge representation in a reusable and machine-readable format. Consequently, ontologies enable the representation and communication of business, network and NEM information, and the development of reasoning schemes. To this effect, the utilization of ontology-based policy translation is suitable for the relevant UMF mechanisms. In order to achieve our goal, the formulation and realization of the ontology should be suitable in order to support taxonomy among the defined concepts and specification of relations in the form of subject-predicate-object (SPO) clauses. Furthermore, the designed ontology should ensure consistency of the included captured concepts and inferring of relations by assignment of meta-properties to existing properties, enabling deductive

and inductive reasoning. For this reason, the selection of Web Ontology Language (OWL) for the ontology foundation and Semantic Web Rule Language (SWRL) for the construction of the conceptual (ontological) rules that enable the information mapping between the different levels (semantic enrichment of the ontology), seems the most appropriate solution.

The translation process adopts the Policy Continuum approach as described in Section 3.3.2, and is accomplished in every policy level from the Policy Derivation and Management Function, through mapping of the policies' parameters to information parameters (and respective attributes) of the ontology. According to this approach, a set of three different levels / views are defined (Business Level, Service Level and NEM Level), each of which constitutes a different representation of the initial business goal. The policies of all levels are described in OWL. The policies in business level are modelled based on the ontology reflecting the business level, close to natural language, while the policies of other levels are modelled on the policy language based on SID information model. The ontologies of different levels are linked in OWL by means of interoperability relationships between classes, which express the interrelation between subsequent levels, while SWRL rules are used for the translation of the policies. The translation is realised by the implementation of translation algorithms expressed in SWRL which transform, generate and deliver data from one level to the subsequent level.

The translation process is illustrated in Figure 35.

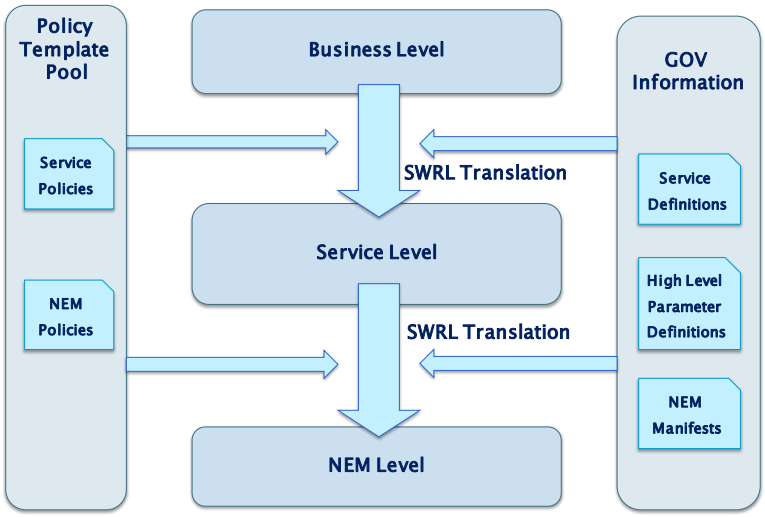


Figure 35: Policy Translation

The translation process comprises the following steps:

Step 1: The initial Policy Variables which constitute High Level Objectives for the operator, are classified to a High Level Objectives' Category (HLO Category) based on operator's selections. For instance, HLO "Availability" (Figure 40) may be comprised of four categories (i.e. Excellent, Good, Normal, Critical). Through the use of Human to Network function, the operator has the possibility to introduce both High Level Objectives and their categories respectively.

Step 2: Based on High Level Objectives Definition, Build Business Policy operation expresses them in the form of business level policies.

Step 3: The initial translation from business policies to service policies is realised based on High Level Objectives' Category to KPIs mapping, the selected combination of high level semantics and the mapping of available service to KPI values (e.g. KPIDelay < 50 msec). For each HLO Category a series of service policy templates are extracted from the policy template pool. The selection of the appropriate policy template is done based on the set of KPIs involved and the initial classification. In general each policy template is a policy skeleton which 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 instances of policies are generated.

Step 4: On the second level of translation each service level policy is further translated into a set of operational NEM level policies. The translation is realised by using a set of NEM templates, including KPI/parameter related

information filling them with KPI/parameter's values. In this step a mapping is realised between the involved KPIs/parameters and the available operations, inputs and outputs of the available NEMs, which are described into the NEM Manifest.

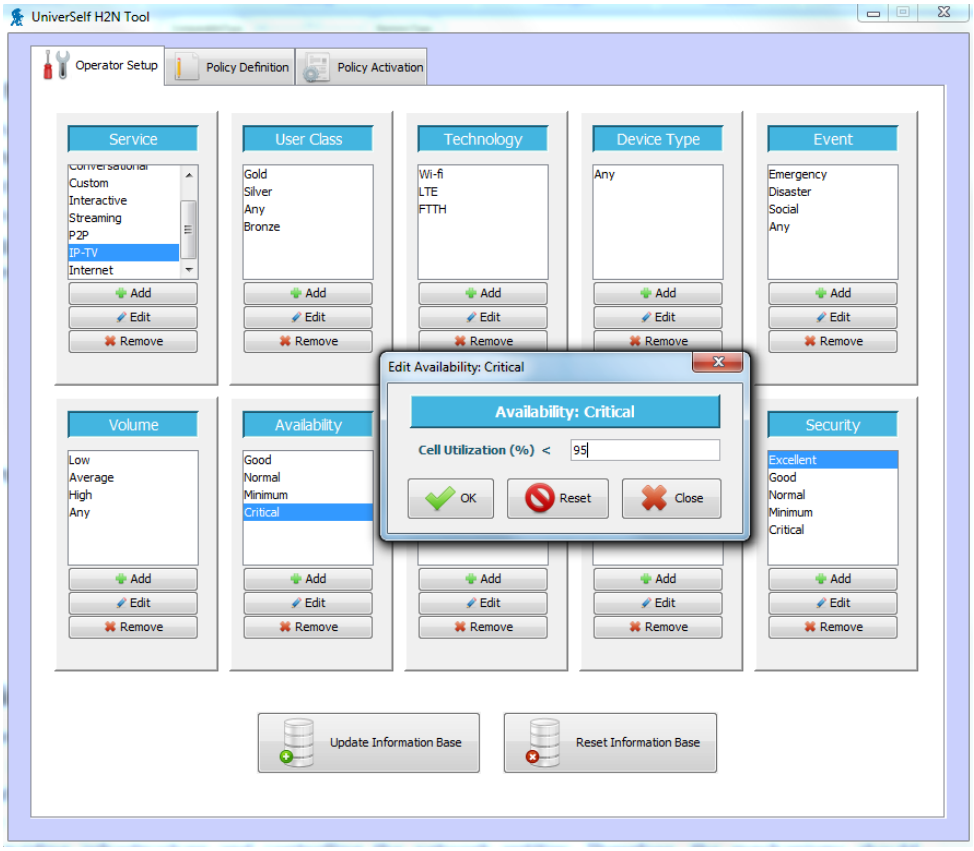


Figure 40: Definition of High Level Objective "Availability" and its categories

In its general form a NEM policy is formed and described based on the SID policy model (see Section 3.3.2).Examples of some indicative policies are:

- Energy Efficient Policy: *IF (time > 3:00 AND time< 8:00 AND NetworkUtilisation < 0.2 AND PercentageOfInactiveENodeBs < 0.1) THEN SET PercentageOfInactiveENodeBs = 0.1*.

  The NEM policy implies that the NEM that will receive the policy from GOV has the capabilities to monitor the network utilization (KPI) and control the percentage of inactive eNodeBs in the network.

- QoS/QoE Policy: ON SLAViolationAlarm IF *SLAClass = Gold AND AdmissionControlThreshold > 0.9 THEN SET AdmissionControlThreshold = 0.9*

  The above NEM policy implies that the specific NEM can receive SLA violations alarms and control the threshold of admission control mechanism of the access network.

4.1.2 Policy conflict detection and resolution

As mentioned in the Policy Derivation and Management function description, Detect Policy Conflicts operation examines policies for conflict at each level of the policy continuum. Specifically, when the number of the policies that exist in the Policy repository in the same level of the continuum is augmented, the Detect Policy Conflicts (DPC) and Resolve Policy Conflicts (RPC) operations are triggered in order to be guaranteed that the active policies, namely the policies that appertain to current time constraints and network/service conditions, in the specific level of policy continuum are conflict-free. These mechanisms ensure that there are not policies in the policy list of the repository of the same level of the continuum that are simultaneously satisfied and the resulted actions from the policies' realization are contradictory.

In this section, two different mechanisms are proposed: in the first one, the conflict detection and resolution processes are interlinked, while the second one is a semantic-based approach for conflict detection only.

Policy Conflict Detection and Resolution mechanism

DPC operation receives as input from the Translate Policy operation a Policy Set object. Figure 41 illustrates the structure of this object, following the SID-based UMF information model. The arrows and numbers in the figure show the relationship among the objects that are included in a policy set object. More specifically a Policy Set object comprises a set of Policy Rules which in turn has three objects that include all the events, the conditions and the actions specified after the policy translation procedure.

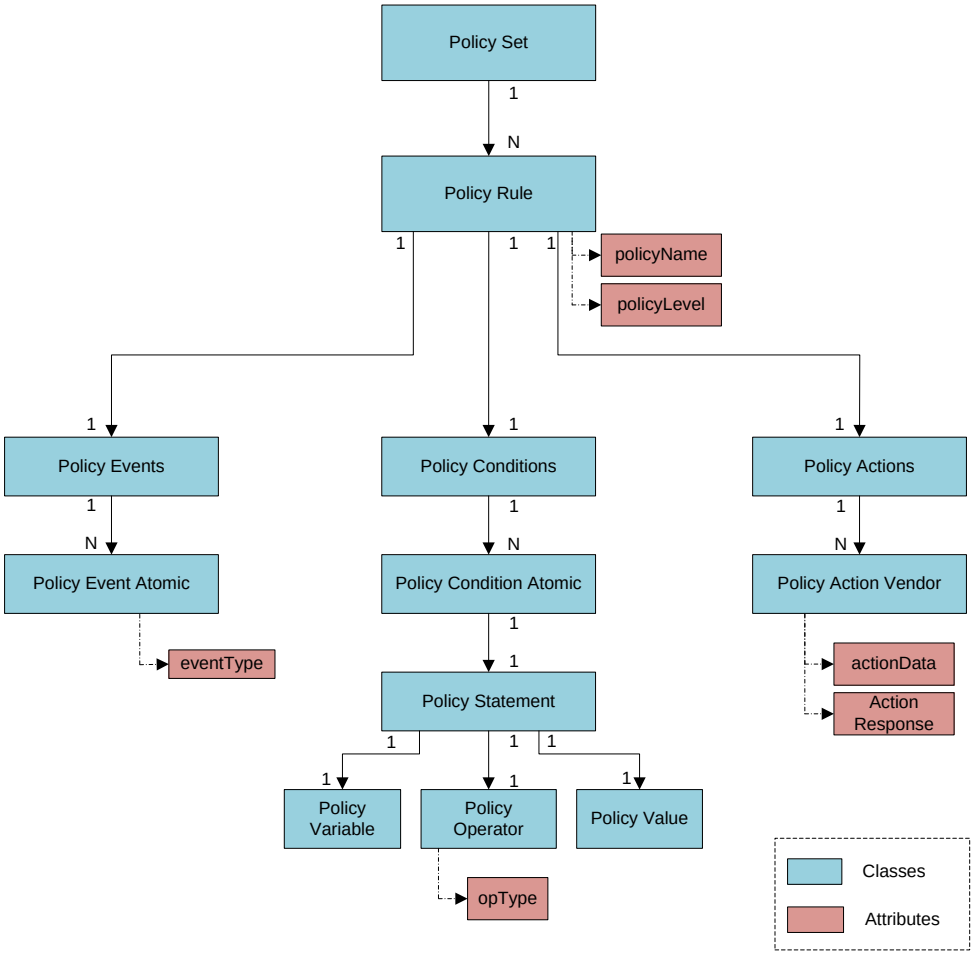


Figure 41: Policy Set object

DPC and RPC operations are executed each time a new policy set is received from the Translate Policy operation. DPC initially identifies whether the rules included in the Policy Set have already been deployed in the system (i.e. after a previous translation process execution) or are newly emerged, and then, splits these rules to two corresponding lists. The identification of already deployed rules can be achieved by keeping a cache, which has been examined for conflict during past executions of the DPC and RPC operations and are conflict-free.

A conflict is defined to occur when two policies can be triggered and satisfied at the same time and where their actions contradict. That implies that the event clauses and the condition clauses of two policies evaluate to TRUE at the same time, but the actions lead to contradictory states. Then, the proposed mechanism identifies potential conflicts among each of the deployed rules with each of the new rules in a three-step process, where each step analyzes one of the three clauses (events, conditions, actions) and filters the set of policies that could lead to a policy conflict.

- Firstly, it analyzes the events of each policy rule trying to identify situations that could lead to policy conflict. This step produces a list of pairs <Deployed Rule i, New Rule j> meaning that Deployed Rule i is conflicting on the event level with New Rule j. Furthermore, two sets of conflict-free sets with policy event atomic objects are produced and are kept separate lists. Those lists will replace the original lists in case the pair of the deployed and the new rule passes through the whole procedure and are indeed conflicting. It is worth mentioning that two cases are identified in this level. The first one is the case where both rules have the same event triggering their actions based on their conditions. In this case we cannot avoid (i.e. refine the events) of these rules and it is mandatory to refine the conditions if the DPC shows that these rules are conflicting. The latter case considers cases where one rule has a Policy Event Atomic while the other one has no event specified. In that case a policy event atomic is produced for the latter rule so as to restrict its occurrence under circumstances that the event of the former rule is not occurred.
- Then, on the second step, for each of the pairs produced during the first step, the mechanism checks whether their conditions could lead to policy conflicts. Such circumstance occurs if and only if the following is valid for all the condition atomics of the rule (i.e. the deployed or the new) with the fewer number of conditions (i.e. policy condition atomics):
    - Assuming that C1 is an atomic policy condition of the deployed rule and C2 is the atomic policy condition of the new rule, then C1.policy variable should be equal to C2.policy variable and their corresponding policy operators and policy values should be, such that a common ground between them exists. For example
        - C1 necessitates that CellUtilization > 0.6
        - C2 necessitates that Cell Utilization <0.8

    For all those pairs received as a list from the first step, that are conflicting in condition level, the corresponding conditions that resolve the conflict are produced and are kept in separate lists. For example for the case of C1 and C2 the conflict-free conditions could be the following:

    1. C1 necessitates that Cell Utilization > 0.6
    2. C2' necessitates that Cell Utilization <= 0.6

    This is an iterative procedure where we check one by one the PolicyConditionAtomics of one Policy Rule against the PolicyConditionAtomics of the other Policy Rule. If the PolicyVariables of the PolicyConditionAtomics of the first Policy Rule exist in the Policy Variables of the Policy Condition Atomics of the other Policy Rule and a common ground exists for ALL these pairs then the pair of the considered Policy Rules is kept for further checking (i.e. check on the action level). Otherwise this pair is not conflicting because either there exist a Policy Variable in one Policy rule that does not appear in the other Policy Rule, or there is a PolicyVariable common in both Policy Rules but with no common ground.
- Then, these pairs of rules pass to the third step that checks conflicts in their actions. In this step a complementary Service Interdependences Matrix is checked to evaluate whether conflicts on action level exist. If so, the conditions of the rules are replaced with the conflict-free ones and the new policy set is composed. This object is the output of the RPC operation. Finally, the cache memory with the deployed rules is updated accordingly.

It should be noted that, given the fact that policy translation mechanism has a deterministic nature (i.e. same business objectives will lead to exactly the same conditions, events and actions) the PCR mechanism described here, will always offer refined solutions that resolve potential conflicts among policy rules.

Semantic-based approach for policy conflict detection

The semantic based approach for the conflict identification described here concerns the identification of parameters and their assigned values in the event, conditions and action clauses that can lead to a conflict between policies. This implies the definition of the suitable classes, properties and axioms (i.e. the TBox). A relevant taxonomy between concepts defined for conflict identification in OWL is illustrated in Figure 42 below. For example, an information piece can be correlated to:

- Parameter (observable and adjustable) comprising ComparableParameters (where predicates like

isLessThan/isGreaterThan can be applied to their values) and NonComparableParameters (where exists no natural ordering among their values, but isEqualTo/isNotEqualTo can still be applied),

- Value/ ValueSet, comprising ComparableValueSet, (which has a lower and upper bound, either exclusive or inclusive) and NonComparableValueSet (which is a set of values, a union, or an intersection of sets)
- ParameterType (which is the data type to be used for assigning actual values)

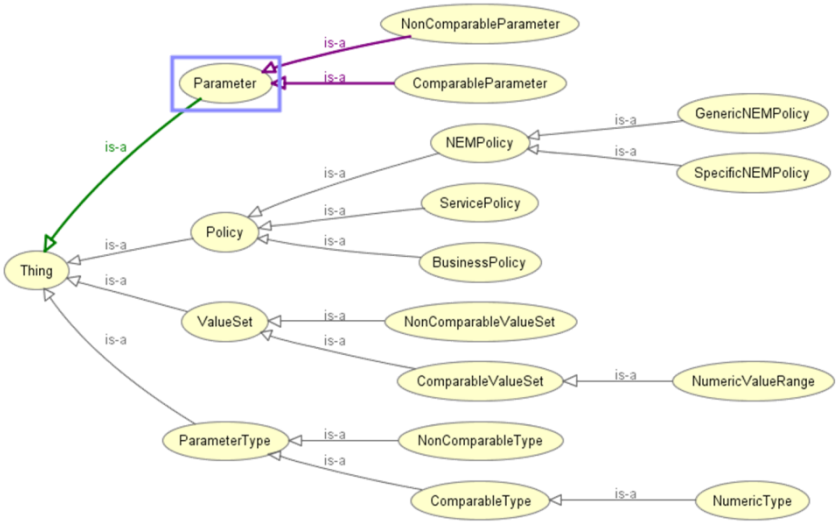


Figure 42: Taxonomy between concepts defined for conflict identification

In the case of the action clauses, the identification of possible conflicts is transformed into detection of the alteration of the targeted parameters from the policies. Then the focus is on the action of the active policies, namely the targeted control of specific parameters. So, the active policies of the same policy level are examined based on the final targeted/influenced parameters. For example,

policyA minimizes parameterX (Goal of policyA),
policyB maximizes (or maintains value of) parameterX (Goal of policyB),

then these policies are identified as conflicted policies. It should be noted that a similar approach can be utilized in evaluation of event and condition clauses of policies.

The PROLOG programming language can be selected for the implementation of the policy conflict detection, as it is a powerful tool that supports non-monotonic reasoning on OWL ontologies (with the assistance of suitable libraries) and interoperability with Java.

It is worth noting that the mechanism described above for the policy conflict identification can also be utilized for the detection of conflicts between NEM actions, operation that is carried out in the Coordination core block. The mechanisms concern the identification of conflicting control-goals over parameters, which can be the outcomes of deployed policies (in Governance) or NEMs actions (in Coordination).

4.1.3 Policy Efficiency 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 Policy Efficiency and measure the gains from the policy application is necessary. The efficiency of a policy in accomplishing the objectives 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 Efficiency**.

In order to estimate Trust, the Entropy-based trust model [11] 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 high level objectives set by the operator. Information theory states that entropy is a nature measure for uncertainty [12]. Thus, entropy-based trust value is defined as:

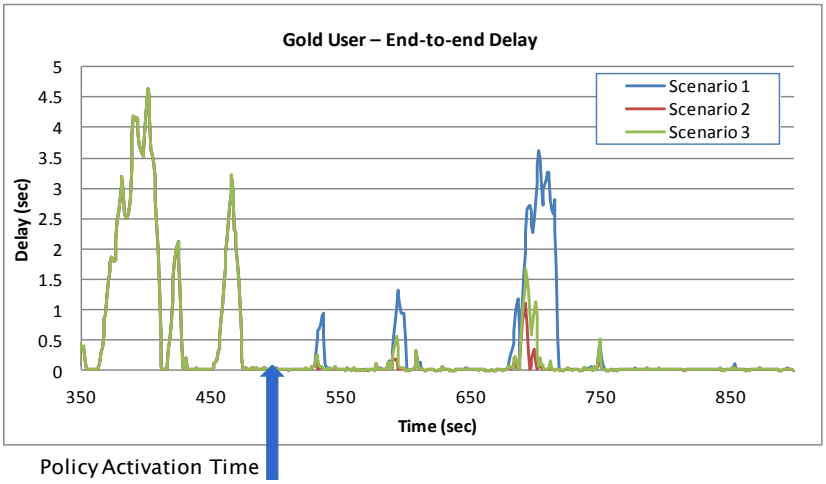
$$T\{subject: policy, successful\} = \begin{cases} 1 - H(p) & 0.5 \le 0 < 1 \\ H(p) - 1 & 0 \le p < 0.5 \end{cases}$$

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 efficiency operation calculates p after the implementation of a policy, based on network measurements collected by agents (reflecting a set of KQIs and KPIs according to [13]) 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 function in order to enable the supervision of the underlying autonomic functioning.

An example of Trust of policy estimation is illustrated in Figure 43. In the considered cases we evaluate three scenarios. In Scenario 1, the operator does not define and activate any high level objective. In this case the network elements retain their initial configuration, which in the scenario means that the WFQ weight is set to 20 for all queues of network routers. In Scenario 2, the operator defines the high level objective “Gold users using the Streaming service should experience Good level of Speed”. According to the proposed methodology, this high level objective is translated to the low level action: "fair-queue qos-group 4 weight 40" which means that after the activation of the policy, the WFQ weight of TOC=4 queue (queue of gold users) will be set to 40. In Scenario 3, the operator defines the same business goal with the previous scenario. It is assumed that in this scenario, the policy translation process results into a different policy representation based on slightly different models and model mapping. The initial high level objective is translated to a device command of changing the WFQ weight to 30: "fair-queue qos-group 4 weight 30". The trustworthiness of the corresponding candidate policies are estimated based on network measurements. In this example, in order for the policies to be evaluated, the following KPI is selected: 90% of the end-to-end packet delay values should be below 200msec. The p value is calculated after the enforcement of the policy, while it is assumed that a policy is successful if the KPI is valid (Trust value > 0). The estimated values of p, H(p) and policy trustworthiness according to the aforementioned methodology are illustrated in the table below. In detail, 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 (where one policy triggers specific low level actions) is more trustworthy than policy of scenario 3(where one policy triggers different low level actions). 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 43: Policy Efficiency estimation based on Trust

4.1.4 Check Feasibility & Optimize mechanism

An important aspect in Governance is the selection of the set of NEMs that have to be involved in handling a situation, namely the NEMs that are candidates to receive new or updated policies because of a change in the network context or an operator's decision. This kind of selection is made by the Check Feasibility & Optimize (CFO) mechanism that is described in this section. In order to succeed in its role, the CFO mechanism requires a variety of input information, related to the features of each NEM, the nature of the request that has been received, as well as the conditions in the network environment. CFO's responsibility at this stage is not to find a fully detailed and specific solution, but rather to outline a feasible and optimized way in which the available NEMs and the corresponding network infrastructure, can handle the request. More specifically, the CFO mechanism shares the request into appropriate pieces and allocates each one of them to a selected NEM. Then it is each NEM's responsibility, as autonomic entity, to act accordingly in order to cope with the requirements that are being posed in the form of policies.

As always, selection implies comparison, which in order to be fair, it has to be done based on the same terms. The NEM manifest definition enables the classification of NEMs based on a number of basic common described attributes. And then the NEM developer should assign a score per attribute that characterizes this NEM. This score can potentially derive from a third party evaluation, in order to increase its reliability. But in any case, the cognitive and knowledge building mechanisms included in the UMF core, are constantly monitoring and assessing these values, and thus they eventually find out the real value of each NEM attribute as it happens to be "in action".

The number of the attributes or the range of their score values is not affecting the essence of the algorithm that resides inside the CFO mechanism; it is built in a generic way, trying to respect the restrictions and to maximize the value of an appropriately formulated objective function. In the following, five main areas have been defined, based on which the selection of NEMs will be made, as presented below.

1. Supported network technologies, services and quality levels: It applies mainly to NEMs that manage network segments, consisted of the underlying infrastructure. Hence it refers probably to the majority of the NEMs in a system. The value is actually a list of the provided network technologies, as well as of the services and the quality levels per service that can be supported, i.e. a RAN NEM controlling some

LTE base stations will normally report a broader QoS level support than a RAN NEM for a set of UMTS Node Bs.

2. Categories of problems that can be handled and solution methods: The nature of the problem is described based on a pre-defined taxonomy, which can actually become known through the UMF information model. The NEM developer specifies the situations in which the NEM can be useful, i.e. Resource Optimization or Knowledge Building. He also provides information on the method that is used to find a solution, as this can be taken into account by the CFO mechanism to assess other attributes, like the next two.

3. Response time: Since every NEM has a known task to accomplish, an estimation of the time needed for that is a criterion that should be taken into account. Probably this has to do with the quantitative characteristics of the problem as well, but the purpose of this attribute is more to give some qualitative indication. So, even a small set of time scale levels could be enough, like i.e. instant, short-term, mid-term, long-term, offline. This information can be provided by the NEM developer and certified by external authorities, being therefore included in the NEM certificate, or it can be the result of the continuous monitoring of the NEM activity.

4. Approximation to the optimum solution: This is a useful input especially in conjunction with the previous one, since i.e. it can justify an increased response time, if it is desired to find a more optimum solution. It may be something measured by independent third party organizations, constituting a trust index resulting from an offline trust evaluation.

5. Energy needs: As it happens with almost all technology products nowadays, the energy class is a fact that has to be known. It has to be noted though, that probably this is a value estimated and reported to the UMF system by the NEM, based on the specific deployment conditions and it cannot be supplied a priori, i.e. a RAN NEM developed for managing LTE base stations, will report different energy needs depending on the underlying infrastructure that has been assigned to it.

An important matter that has to be highlighted, is that the UMF core is using its own monitoring and assessment mechanisms, the so-called online trust evaluation mechanisms (see Section 0), in order to maintain a general trust index for each NEM, depicting how consistent are the scores in each attribute with the reality. This is the “defence” of the system towards NEMs and developers that have (intentionally or not) over-estimated their capabilities or have potentially overlooked important aspects that affect its behaviour and operation. Using the trust index, the CFO mechanism can promote or put aside a NEM under specific circumstances.

Other information related to NEM operation that is considered for its involvement or not into the handling of a request, is the geographical area where it is deployed, which has to sufficiently overlap with this of the request. And besides that, the status and the availability of the NEM are also checked. At the end, the CFO mechanism has selected the NEMs that will handle the request, as well as the responsibilities that they can undertake and it adapts the issued policies according to this decision.

For each NEM, a specific value is assigned to each of the aforementioned criteria. All the values lie within the same range in order to be comparable. Additionally, each value is multiplied by a priority factor (from 0 to 1), which depicts the significance of a criterion in a certain problem. The sum of these products it is multiplied with the NEM’s trust index and this is how it is derived the suitability of a NEM for contributing to the problem’s solution. A suitability value equal to zero indicates that a NEM cannot be selected, since i.e. it operates in a totally different area or it is not able to deal with problems like the one in question etc.

Thus the selection process has 3 main phases. The first one is a filtering of all the registered NEMs in order to find the NEMs that are candidates to contribute to the problem solution. This is done on the basis of information stemming from the NEMs configuration, like the geographical location of the infrastructure that this NEM controls and the first two of the criteria mentioned above. The second phase includes the computation of the suitability value for each candidate NEM, based on the criteria 3, 4 and 5 and the system’s knowledge that is provided through the trust index. Finally, in the third phase, the most suitable NEMs are selected and an appropriate portion of the problem is allocated to each one of them. The three phases are depicted graphically in Figure 44 below.

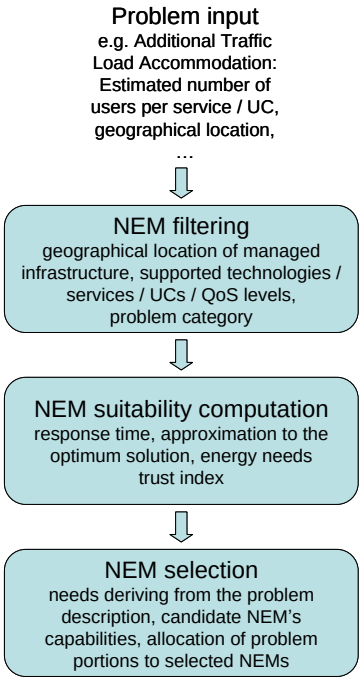


Figure 44: Phases of the NEM selection process

4.2 Information and knowledge management mechanisms

4.2.1 Knowledge production mechanisms

The UMF Ontology as an enabling technology for knowledge production

The UMF ontology aims at providing a dynamic knowledge base capturing the knowledge produced by corresponding mechanisms (i.e. integrated in the knowledge production operation of the IPKP function or in the knowledge producing NEMs) within the different domains of the UMF reference system and integrating/aggregating them in an efficient way. In this sense, KNOW, COORD and GOV viewpoints can be integrated into a single but modular and extensible ontology able to capture additional concepts and interrelations. Moreover, the ontology enables to the KNOW block reasoning and inference capabilities through the definition and incorporation of corresponding rules reflecting the served purposes of the KNOW block. Last but not least, ontology aims at capturing the interrelations in a way which can be utilised within UMF. The selected approach aims at (i) developing a generic ontology capturing the key concepts at the higher level of abstraction and (ii) enabling development of "local" ontologies reflecting more specific concepts through specialising the generic ontology concepts. The different levels of abstraction supported are: (i) Network domain level - Core, Access, etc, (ii) Network element level - Router, Base Station, etc, (iii) Network resource level - Link, Channel, Port, etc, and (iv) NEM level - problems, solutions.

For the UMF ontology development, we revisited the specifications of the knowledge production capabilities, which are supported in the form of Knowledge Building NEMs and generalized, in order to provide general-purpose knowledge building capabilities integrated in the KNOW, i.e., the IPKP function specified in the previous subsection. The following tables present the Knowledge Building NEMs we used for the ontology development. In this process, we considered the fields: (i) problem to address, (ii) the knowledge production mechanisms used to tackle the problem, (iii) inputs for the knowledge production, (iv) inputs location, (v) output knowledge, and (vi) output location.

NEM	**MPLS Traffic Engineering**
Problem to address:	Avoid core network load/congestion
Solution to the problem:	Evolutionary Techniques exploited to minimize (i.e. balance/split) the maximum load that passes from the network links.
Inputs of knowledge production:	▪ Total capacity, ▪ Current Load, ▪ Port Descriptors, ▪ Incoming and Outgoing Flows, ▪ Source/Destination IPs and flow demands between them, Equipment ID, ▪ Port capacity and status (associated to a link or not), ▪ Costs associated to router links
Inputs location:	UMF KNOW (or monitoring entities directly)
Outputs:	Update Routing table (change metrics)
Outputs location:	UMF KNOW

NEM	**Cell Outage Compensation (Self-Healing)**
Problem to address:	This NEM aims to solve the emergency case of an access point (AP) failure by adjusting the transmission power of the neighbouring APs.
Solution to the problem:	Fuzzy Inference Engine
Inputs of knowledge production:	List of RAN elements in the form of: < RAN Element id, current TxPower (dBm), Load level, channel, list of associated terminals <UE id, bit rate, RSS>> RAN Element id of AP in outage
Inputs location:	UMF KNOW (or monitoring entities directly)
Outputs:	Updated list of compensated RAN elements in the form of: < RAN Element id, updated TxPower (dBm), updated Load level> List of reconnected clients <UE id, interference values>
Outputs location:	UMF KNOW

NEM	**Congestion Prediction**
Problem to address:	The problem addressed is the empowering of the system to combine monitored network data in order to predict a possible congestion on any specific link.
Solution to the problem:	Unsupervised Machine Learning: Parameterless Growing Self-Organizing Map (PLGSOM)
Inputs of knowledge production:	Periodic measurements of: ▪ incoming traffic ▪ in packets ▪ in bytes ▪ trend of incoming traffic ▪ queue load ▪ in packets ▪ link capacity ▪ buffer size or, (pre-processed) ▪ link utilization (instead of incoming traffic and link capacity) ▪ queue utilization (instead of queue load, buffer size)

	▪ trend of link utilization (instead of trend of incoming traffic) together with: ▪ a congestion level label (not congested, towards congestion, congested) for each measurement
Inputs location:	UMF KNOW (or monitoring entities directly)
Outputs:	congestion level label for a given context described by the aforementioned parameters
Outputs location:	UMF KNOW

NEM	**Load Level Estimation (LLE)**
Problem to address:	Provide estimations to other mechanisms regarding traffic load for RAN elements, in specific time periods.
Solution to the problem:	Unsupervised Machine Learning: Parameterless Growing Self-Organizing Map (PLGSOM)
Inputs of knowledge production:	▪ Time of measurement ▪ Load level ▪ RAN Element id
Inputs location:	UMF KNOW (or monitoring entities directly)
Outputs:	Load estimations in the form: < RAN Element id, List<Pair<Load Level, Probability/Percentage>>>
Outputs location:	UMF KNOW

NEM	**Building Knowledge on the Trustworthiness of a Control Loop/ set of control loops (Trust mechanism)**
Problem to address:	In order to enforce business goals or decisions, there is the need to assess if the decision done by one or a set of control loops (denoted by monitored NEMS) is trustworthy, in other words that the taken decision is good with respect to the business goals. This mechanism is building and providing the necessary knowledge that would help ▪ to influence the decision of monitored NEMs, helping them to make good decisions according to the business goals ▪ to identify inside a family of similar control loops (making the same kind of decisions based on various algorithms) the ones that are providing the better behaviour (according to business goals).
Solution to the problem:	Reinforcement Learning: Q-Learning
Inputs of knowledge production:	▪ Context information (parameters comprised in the trigger of the control loop(s)) ▪ Decision taken from the control loop ▪ KPIs/ Metrics that can be used (depending on the control loop) for defining their deviation from the requested goals, e.g., ○ Quality of Service (QoS), ○ Number of resources involved for the enforcement of certain governance and self-management actions, ○ Time required for the enforcement of certain governance and self-management actions, ○ Number of reconfigurations deriving from certain governance and self-management actions.
Inputs location:	UMF KNOW (supposing that the under trust NEMs and monitoring entities will already have sent this information to the KNOW block)

Outputs:	▪ Which decision of a control loop is the most trustworthy, given the current state of the system, ▪ Which control loop is the most trustworthy to reach a decision.
Outputs location:	UMF KNOW

Based on the above inputs, a generic ontology has been identified encompassing the following concepts based on the considered levels of abstraction: (i) Network domain - i.e. core, RAN, (ii) Network element - i.e. router, link, AP, etc, (iii) Profile (i.e. static information) and measurements (i.e. dynamic information) which can be subject to monitoring and are further broken down into attributes and parameters, (iv) Problem abstracting - the problem addressed by the specific knowledge building core mechanism, (v) Solution - abstracting the available solutions linked to problems, and (vi) Knowledge - abstracting the knowledge produced by the mechanisms realizing the solutions, linked to solutions.

Figure 4 presents the UMF ontology capturing the following concepts, problems, solutions and produced knowledge:

- Network congestion
  - *isA* Problem
  - *hasDomain* Core
  - Solution-1: *hasSolution*MPLS Traffic Engineering
  - Produced knowledge
    - *hasKnowledge*Updated routing table
  - Solution 2: *hasSolution* Congestion prediction
  - Produced knowledge
    - *hasKnowledge*Predictions on congestion level per network element
    - *hasKnowledge*Probability
- Cell outage
  - *isA* Problem
  - *hasDomain* RAN
  - Solution: *hasSolution* Cell outage compensation
  - Produced knowledge
    - *hasKnowledge*Compensated RAN elements
    - *hasKnowledge*Reconnected users
    - *hasKnowledge*Interference
- RAN load
  - *IsA* Problem
  - hasDomainRAN
  - Solution: *hasSolution* Load level estimation
  - Produced Knowledge:
    - *hasKnowledge* Load level estimation
    - *hasKnowledge* Probability.

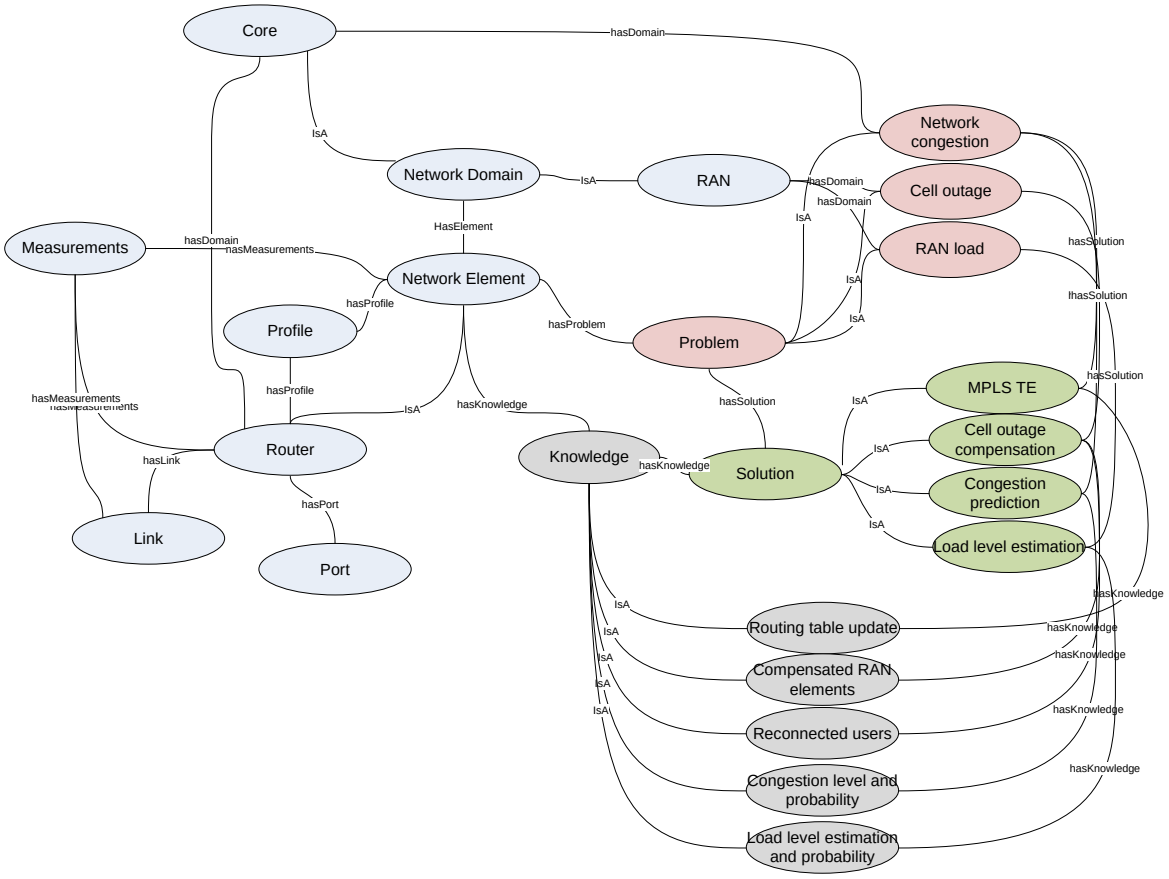


Figure 45. Generic ontology

In this sense, the generic ontology provides knowledge traceable as follows: Network congestion *IsA* Problem which can occur within Core which *IsA* Network Domain that includes a number of Routers which *IsA* Network Element. Network Congestion *hasSolution* provided by either the MPLS TE which *IsA* Solution and *hasKnowledge* related to Routing table update, or, *hasSolution* provided by the Congestion prediction which *IsA* Solution and *hasKnowledge* related to predicted congestion level and respective probability.

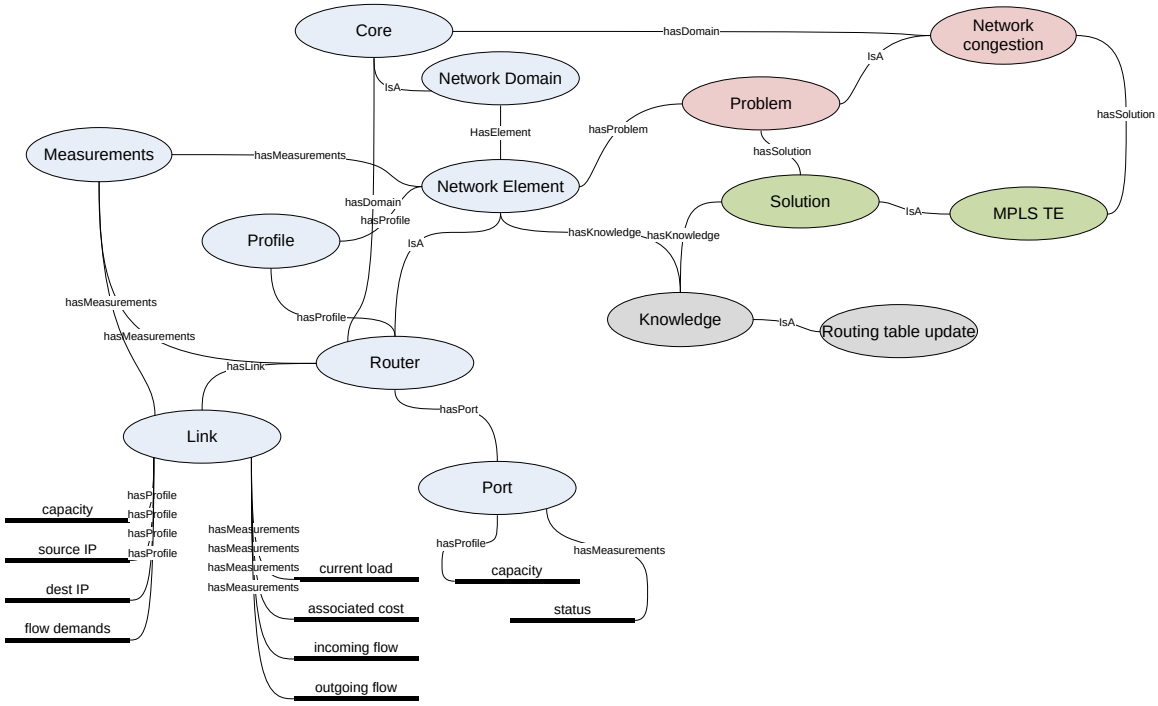


Figure 46. MPLS TE - knowledge production mechanism.

The above Figure 46 presents a more specific ontology capturing the concepts related to the MPLS TE production mechanism. Such concepts are incorporated within the ontology as further breaking down the profile and measurements concepts and are interrelated to network elements.

In this example mechanism the Link and the Port are involved and their profile and measurements are included in the ontology as presented in the following table.

Concept	Profile	Measurements
Link	Capacity, Source IP, Dest IP and Flow demands	Current load, Associated cost, Incoming flow and Outgoing flow
Port	Capacity	Status

Semantic based-Knowledge production mechanisms

In this section, we present how Semantics and Ontology-based technologies can represent knowledge. The main functionality is based on an ontology constructed in Ontology Web Language (OWL).

An ontology enables the capture and organization of our knowledge about the world in a reusable and machine-readable format. In particular, it provides a taxonomy among well-defined concepts (or classes). Namely, classes are organized into a subclass-superclass hierarchy. So, we can consider that subclasses are subsumed by superclasses. In our case, classes would be 'Router', 'NetworkElement', 'NetworkDevice', 'NetworkInterface' etc and we say that 'Router' is subsumed by 'NetworkElement' or alternatively that 'Router' is-a 'NetworkElement' (see Figure 47).

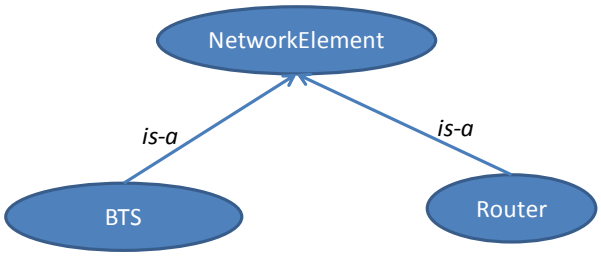


Figure 47: Taxonomy in OWL

Figure 48: SPO notion

Each class has instances or objects which are called individuals. Individuals are also referred to as being instances of classes. For example, as depicted in Figure 48, individual 'ALU-SAR7705#15.3' is an instance of class 'Router'. In an ontology, the individuals would have relations (predicates) linking to each other, and thus forming Subject-Predicate-Object (SPO) clauses, where both "subject" and "object" are individuals (also depicted in Figure 48). For example, a predicate (a.k.a. object or data property) would be "isConnectedTo", "hasAddress", "hasPsysicalAddress", etc.

An ontology enables the description and therefore, the capture of real world-, as well as system-knowledge of the UMF domain. Such knowledge is often acquired by network experts but is mostly not machine-exploitable. Furthermore, it can support several UMF tasks such as the identification and resolution of conflicts as well as the policy translation proccess. Moreover, given an ontology, further knowledge can be produced at run-time by classification and inference methods.

Reasoning is the main process that could be applied on an ontology in order to produce and maintain knowledge. The software component implementing such functionalities is called a "reasoner" (a.k.a. classifier). The main features of a reasoner are summarized in the following:

- Ensure consistency of the captured concepts;
- Infers relations between individuals by analyzing the assigned meta-properties (these of the meta-model) to the properties of the model;
- Classifies individuals and infers new relations based on axioms and rules (deductive reasoning).

Assurance of the consistency of the captured concepts

A reasoner provides consistency checking. Based on the description (conditions) of a class (concept) the reasoner can check whether or not it is possible for the class to have any instances. A class is deemed to be inconsistent if it cannot possibly have any instances.

Inference of relations by analysis of the meta-properties assigned to existing properties

An object property in an ontology (relation among individuals) may be semantically enriched by any meta-property (or axiom) provided by the meta-model (i.e. the ontology language itself). A short description of such meta-properties provided by OWL, follows:

- Inverse Property: If some object property links an individual A to an individual B, then its inverse property will link the individual B to the individual A. For example, let's assume that the object property 'hasAddress' links the individual 'eth0' to the individual '134.80.56.101'. Next, we assume that 'hasAddress' has as inverse property, e.g., the 'isAddressOf'. So, we infer that object property 'isAddressOf' links the individual '134.80.56.101' to the individual 'eth0' (see also Figure 45).

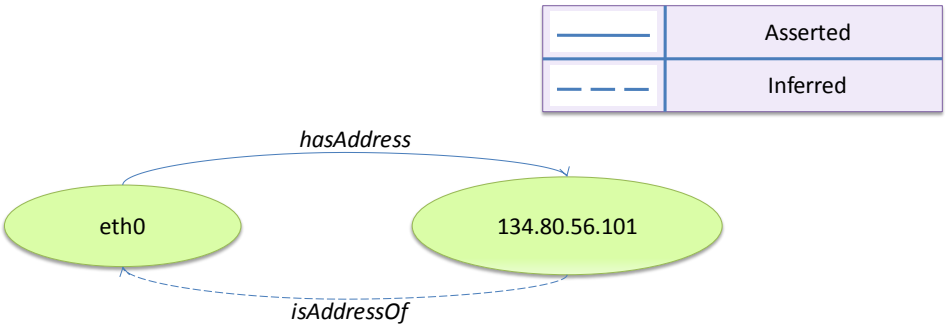


Figure 45: Inverse Property

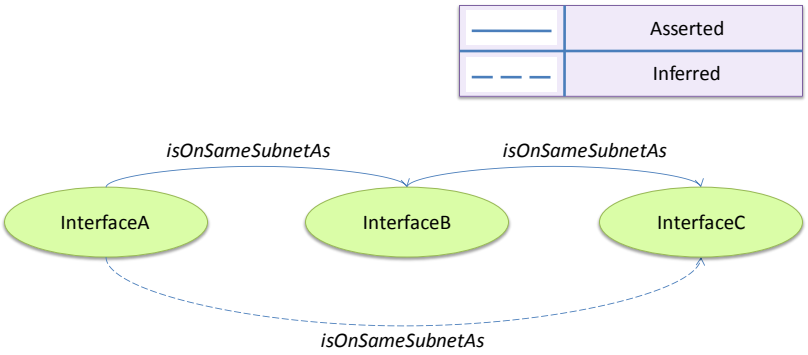


Figure 50: Transitive Property

- Functional Property: A Functional property means that an individual A can have only one (unique) value Y. For example, we assume that the property 'hasAddress' is a Functional property. So, each of the individuals of class 'NetworkInterface' can have at most one link to an individual of class 'Address'. Generally, the reasoner checks the consistency by examining if an individual is linked to at most one individual. If not, it alarms an inconsistency in the ontology.

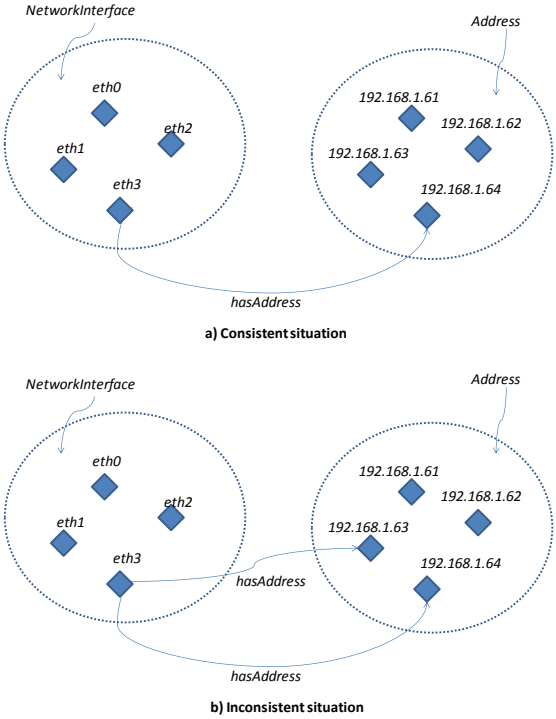


Figure 51: Functional Property

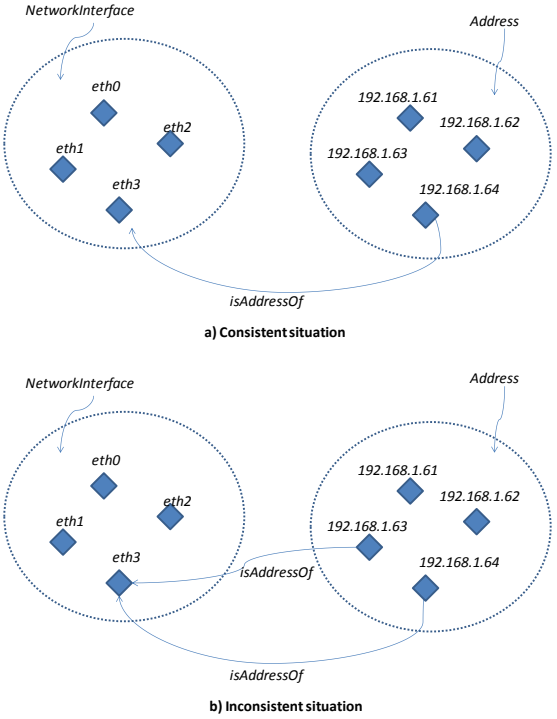


Figure 52: Inverse Functional Property

- Inverse Functional Property: This property is the inverse property of Functional property. In this case, it means that for a given individual, there can be at most one individual related to that individual via this property. Assuming that the property 'isAddressOf' is the inverse property of 'hasAddress' functional property, 'isAddressOf' is its corresponding inverse functional property. So, for example, declare the individual '192.168.1.64' can be declared to be the address of at most one individual of the class 'NetworkInterface'. Otherwise, it is an inconsistent situation (depicted Figure 52).

Transitive Property: We assume that a property 'isOnSameSubnetAs' is declared as Transitive. We assert that an assert that an individual 'InterfaceA' is linked to an individual 'InterfaceB' with the property 'isOnSameSubnetAs' 'isOnSameSubnetAs' and the individual 'InterfaceB' is linked to an individual 'InterfaceC' with the property property 'isOnSameSubnetAs'. Then the reasoner infers that the individual 'InterfaceA' is linked to the individual individual 'InterfaceC' with the property 'isOnSameSubnetAs' (see

- Figure 50).
- Symmetric & Asymmetric Property: We assume that property 'isConnectedTo' is declared as symmetric. If an individual 'InterfaceA' is linked to an individual 'InterfaceB' with property 'isConnectedTo', then the individual 'InterfaceB' is also linked to the individual 'InterfaceA' with the property 'isConnectedTo' (see Figure 53a). There is, also, the asymmetric property which means that if an individual 'NEMx' is linked to an individual 'NEMy' with the asymmetric property 'isMasterOf' then the opposite relation is not possible (see Figure 53b).
- Reflexive & Irreflexive Property: A property is denoted as Reflexive property when the property must relate individual 'IPnodeA' to itself (see Figure 54). On the other hand, a property 'hasConflictWith' is stated as Irreflexive property when we want to relate individual 'NEMx' and individual 'NEMy' with P and the two individuals are completely different. So, in this case the property 'hasConflictWith' cannot relate individual 'NEMx' itself (see Figure 54b).

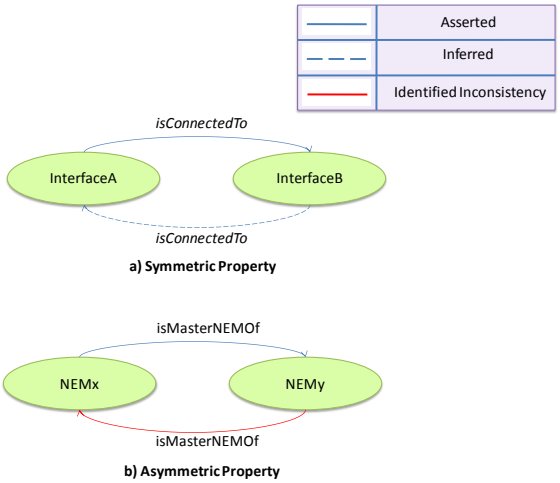


Figure 53: Symmetric and Asymmetric Property

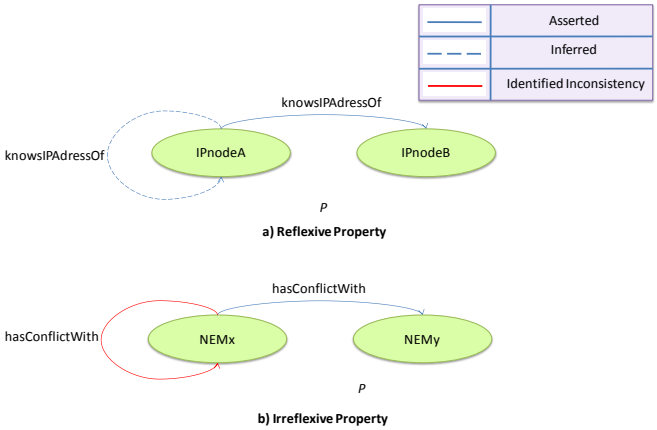


Figure 54: Reflexive & Irreflexive Property

Another issue that OWL technology deploys is the disjointness of classes. When we talk about two classes that are disjointed we mean that these two classes have nothing in common and we cannot relate them in any way. Furthermore, OWL introduces the *existential* and *universal restrictions*. Existential restrictions specify the existence of *at least one* relationship along a given property to an individual that is a member of a specific class. On the other hand, universal restrictions constrain the relationships along a given property to individuals that are members of a specific class.

Essentially, a reasoner has to take into account all the above axioms/properties, if they have been assigned onto classes, and restrictions in order to check the consistency of the ontology and infer all the relations between them. An example that shows the action of a reasoner is depicted in Figure 55.

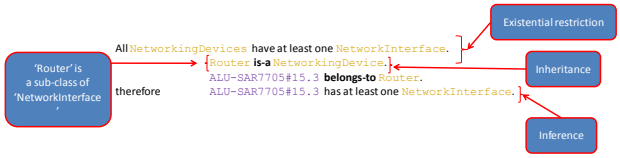


Figure 55: Example of Inference

SharedMedium(?m), Interface(?i1), Interface(?i2), isConnectedTo(?i1, ?m), isConnectedTo(?i2, ?m) -> isConnectedTo(?i1, ?i2)

Figure 56: Example of an SWRL Rule for inferring that interfaces connected to the same shared medium are also connected to each other

Classification and inference based on axioms and rules (deductive reasoning)

Another capability of the OWL technology with reasoners is the classification through the combination of axioms and rules. Classification is a more efficient way to structure knowledge in ontology as the rules give the capability to the reasoner to infer a relation between individuals with which an individual has relations. Rules can also classify new inserted individuals in the ontology.

We can use the Semantic Web Rule Language (SWRL), as a rule language, in order to design and construct the rules. SWRL rules are similar to rules in Prolog. An example of a rule for inferring connection of interfaces that are connected in the same shared mediums is depicted in Figure 56. Comma delimiters (,) represent logical conjunction between the clauses they separate, while the arrow (->) separates the *antecedent* (body) between the consequent (head). Question marks (?) preceding any name denote variables whose values range over individuals of the populated ontology, e.g. for "SharedMedium(?m)", the variable "?m" will be holding one-by-one all individuals belonging to the class "SharedMedium". The rule is interpreted as: *if the antecedent holds (is "true"), then the consequent also holds*. The example illustrates the power of rule languages as a way of capturing the real world knowledge (in this case, the semantics of a shared medium) humans possess for automated use by an expert system.

Data mining-based knowledge production mechanisms

In this section we describe in a theoretical/ generic point of view the data-mining-based knowledge production mechanisms, one of the main options for the knowledge production operation of the IPKP function of the

KNOW block. Knowledge production is defined as a procedure, which directly processes and acts upon raw data within the operator's ecosystem, stemming either from control, service, user or network sources. By exploiting the merits of Data Mining and Machine learning, a significant piece of information can be extracted by such massive datasets thus enhancing operators' management activities as well as the perceived QoS of the end-users. Specifically, data clustering feature extraction and selection, statistical analysis, decision trees and root cause analysis are some indicative techniques for extracting context from merely used data either proactively or reactively. This context may be related to event prediction (i.e. load prediction/estimation, QoS degradation identification) or fault prediction/identification

Unsupervised Learning Mechanisms - Self-Organizing Maps (SOM)

This section presents how Self-Organizing Maps, an unsupervised machine learning technique, can serve as a basis for developing knowledge production mechanisms in the context of the Knowledge Production (KP) operation of the Information Processing and Knowledge Production (IPKP) sub-function. The technique is depicted in Figure 57. To begin with, SOM is a neural network-based approach while the specific version used here, namely the Parameterless Growing SOM (PLGSOM) is a growing variant of SOM equipped with automatic tuning schemes of basic training parameters. Training of SOM is the shaping of a so-called map, which is a 2-dimensional representation of high-dimensional input data. In the training phase, input samples (in the form of vectors that describe the state of the system) are mapped onto the 2D grid in a competitive manner, a process also known as vector quantization. The additional growth process aims at overcoming the traditional SOM restriction for fixed-size maps, while the automatic tuning of parameters allows for additional learning over time. The created map reveals/ depicts the patterns of the past experience of the system and is practically the knowledge base that will be exploited later on to infer/ derive conclusions on future system states/behavior. The mapping and the classification, according to the map, of the new input vectors, that describe partially the future state of the system, are the basis for inferring the missing information and foreseeing the future behavior of the system.

The main algorithm of SOM is used to solve classification problems, which implies, among others, that there exists a set of classes L, in which input data are clustered/ classified (each input sample belongs to one of the classes). The term "label" in this case refers to the value specifying the class in which a vector belongs. For every classification problem, there exists an unknown function $f:\Re^n \mapsto L$, where $\Re^n$ is the input space, and L is the set of discrete classes. Given a set of sample vectors $x_i \in \Re^n$ and their corresponding labels $f(x_i) \in L$, the problem is to find an estimation f˜ for the unknown f.

In order for the Knowledge Production mechanism to build a knowledge base (i.e. the map, which will later be exploited in order to approximate f˜), a set of mappings of the form $\{x_i, f(x_i)\}$, i.e., pairs of n-dimensional input vectors with their corresponding classes needs to be provided. Such an input can also be provided online for real-time knowledge production and adaptation to the input space. For producing the knowledge, the mechanism may use a) monitored raw data, b) aggregated information or c) locally-scoped knowledge from Knowledge Building NEMs of the system depending on the exact problems it deals with. It should be clarified here that the presented technique can be applied for developing a mechanism that produces knowledge and infers future behavior of the system but the exact input parameters that should be used depend on the exact nature of the system behavior one needs to foresee and each case has to be studied separately.

As soon as the map is initiated, the Knowledge Production mechanism is capable of estimating f˜(x_j) for any given input $x_j \in \Re^n$. Moreover, the mechanism provides a certainty indicator of the estimation. Accuracy of the results will be (among others) subject to the number and the quality of the classification of the labelled input vectors provided during the training phase.

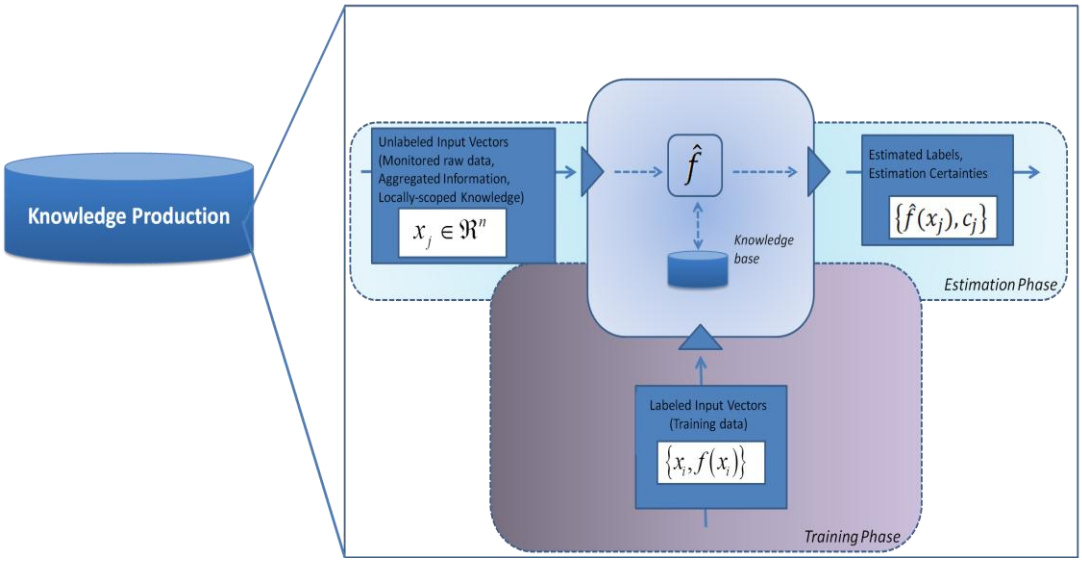


Figure 57: Knowledge Production Approach

Data Pre-processing and Clustering

This section introduces a set of data pre-processing and clustering techniques that are supported from the Information Pre-Processing component of the Information Processing & Knowledge Production (IPKP) sub-function (i.e., see section 3.3.4).

Data normalization

Normalization is a "scaling down" transformation of the features. Within a feature there is often a large difference between the maximum and minimum values, e.g. 0.01 and 1000. When normalization is performed the value magnitudes are scaled to appreciably low values X. The most well-known method for this scope is Min-Max normalization, which is analyzed as follows:

$$v' = \frac{v - \min_A}{\max_A - \min_A} \times (new\_max_A - new\_min_A) + new\_min_A \text{(Error! Bookmark not defined.)}$$

The Kernel Trick

In machine learning, the kernel trick is used to describe the situation where the quality of a linear classifier is enhanced by artificially augmenting the number of dimensions X. Essentially, any kernel function in R^k is a dot-product function in R^n with k<n. Simply stated, such a function projects data in a higher dimensional space where data relations are clearer and linearly separable. One of the most classical kernel functions is the polynomial kernel, defined as $<x, y>^p$, where pЄN.

Data Clustering

The core idea of Data Clustering is to partition a set of N, d-dimensional, observations into groups such that intra-group observations exhibit minimum distances (for example Euclidean distance) from each other, while inter-group distances are maximized. One of the most famous Data Clustering algorithms is k-Means this algorithm will be the core process of the proposed mechanism, thus it will be presented more analytically. K-Means tries to minimize the following objective function:

$$\mathbf{J} = \sum_{\mathbf{i=1}}^{\mathbf{c}} \mathbf{J_i} = \sum_{\mathbf{i=1}}^{\mathbf{c}} \left(\sum_{\mathbf{k}, \mathbf{x_k} \in \mathbf{G_i}} \|\mathbf{x_k} - \mathbf{c_i}\| \right) \quad \textbf{(1)}$$

Where c is the number of clusters, G_i is the i-th group, x_k is the k-th vector in group J_i and represent the Euclidean distance between x_k and the cluster centre c_i. The partitioned groups are defined by using a membership matrix described by the variable U. Each element U_{ij} of this matrix is equal to 1 if the specific j-th data point x_j belongs to cluster i, and 0 otherwise. The element U_{ij} is analyzed as follows:

$$\mathbf{U_{ij}} = \begin{cases} \mathbf{1}, & \textbf{if } \|\mathbf{x_j} - \mathbf{c_i}\|^2 \leq \|\mathbf{x_j} - \mathbf{c_k}\|^2, \textbf{ for each } \mathbf{k} \neq \mathbf{i} \\ \mathbf{0}, & \textbf{otherwise} \end{cases} \quad \textbf{(2)}$$

This means that x_j belongs to group i, if ci is the closest of all centres. The optimal centre that minimizes equation (2) is given below:

$$\mathbf{C_i} = \frac{1}{|\mathbf{G_i}|} \times \sum_{\mathbf{k},\, \mathbf{x_k} \in \mathbf{G_i}} \mathbf{x_k} \quad (3)$$

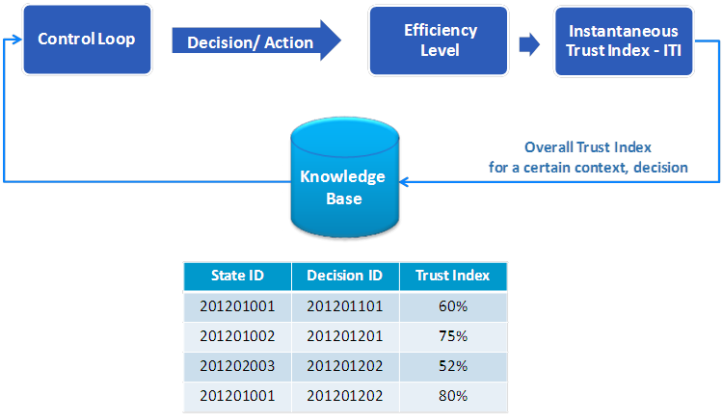
$$|\mathbf{G_i}| = \sum_{\mathbf{j}=1}^{\mathbf{n}} \mathbf{U_{ij}} \quad (4)$$

The produced clusters provide a classification (i.e. label) of the input datasets which can reveal the state of a network element or service (e.g. Load level, QoS level) which can be stored either locally or in a broader scope in order to enable other Self-* capacities within the operators ecosystem.

Trust evaluation and Certification mechanisms

In order to efficiently enforce decisions, there is the need to assess if the decision made by one or a set of control loops (denoted by monitored NEMS) is trustworthy, in other words that the decision made is in accordance to the business goals set, is enforced efficiently and does not compromise the overall operation of the system. This mechanism is building and providing the necessary knowledge related to the efficiency of decision made that can help

- to influence the decision of monitored NEMs, helping them improve the decisions made according to the business goals (Figure 58); and
- to identify inside a family of similar control loops/ NEMs (making the same kind of decisions based on various algorithms) the ones that exhibit the optimal performance, given a certain context (according to business goals) (Figure 59).



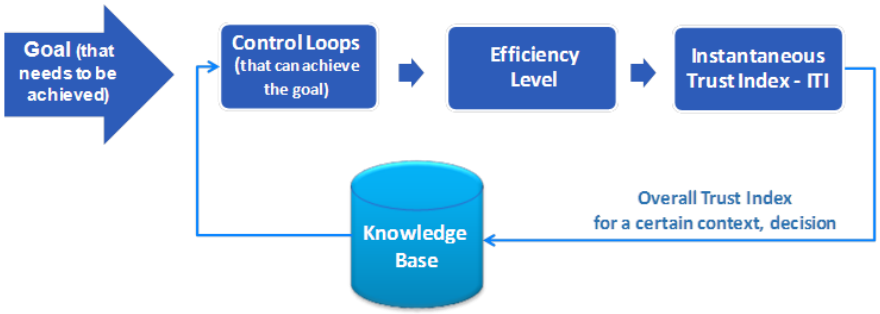
State ID	Decision ID	Trust Index
201201001	201201101	60%
201201002	201201201	75%
201202003	201201202	52%
201201001	201201202	80%

Figure 58. Trust Evaluation of the decisions/ actions of a given NEM.

The proposed mechanism follows a reinforcement learning approach, namely Q-learning in accordance to the description provided in sections 3.2 and 3.4 of [1]. In this approach, the evaluation of trust starts with the identification of the metrics, i.e., generic observable parameters, that can be used for each NEM so as to form the Efficiency Level (EL) of the under evaluation NEM. Those may be (but are not limited to) the following:

- Deviation from requested goals of a control loop (e.g. Quality of Service (QoS) levels)
- Resources involved for the enforcement of certain control loop(s) decisions/actions.
- Time required for the enforcement of control loop decisions/actions.
- Number of reconfigurations deriving from certain control loop decisions/actions.

It should be noted here that different observable parameters may be considered for diverse self-management functionalities/ NEMs (control loops). According to those metrics, the EL, the Instantaneous Trust Index (ITI) and the Overall Trust Index (OTI) are calculated though the mathematics of the Q-learning formulation.



State ID	Control Loop ID	Trust Index
201201001	201201101	60%
201201002	201201201	75%
201202003	201201202	52%
201201001	201201202	80%

Figure 59. Trust Evaluation of the NEM of the same NEM family.

Given the fact that the metrics may differ from NEM to NEM, the mechanism needs to receive the type of the parameters it will take into account from the NEM manifest hold in the registry of the block. Moreover, in order to use the observed value of these parameters and to calculate the EL.

Accordingly, based on the followed approach, for calculating the ITI the mechanism will need the lowest acceptable values of the metrics given the existing high level objectives. These values are be provided by the GOV block with the NEM families/ instances that should be evaluated. This kind of information could be given in an initialization message of the mechanism from GOV.

Last, but not least, the mechanism stores the calculated/ updated OTI given the state the network was before the decision/ action taken by the control loop/ NEM (State ID) and the decision (1st case – Decision ID) or the used NEM (2nd case – Control Loop ID). The state ID is described by context information that need to be fed to the trust mechanism with the decision/ action and the NEM that made it.

Table 38. Overview of the Information to be provided to the mechanism.

Source	Type of Information
NEM manifest/ registry of the block	Parameters that play the role of metrics
KNOW	Observed values
GOV	• Which NEM family/ instances to monitor • Targeted values given the business goal
Under question NEM	• Context Information • Decision/ action

4.3 Coordination mechanisms

4.3.1 Conflict identification mechanisms

Static conflict identification

Static conflict identification is based on apriori knowledge of conflicting parameters that is captured by the UMF system (mostly offline) and is then exploited to identify conflicts at runtime. In contrast to dynamic conflict identification, which can take place even without such knowledge, the static counterpart, is triggered mostly upon configuration changes in the network or the UMF (e.g. the assignment of a mandate to a NEM), in order to detect and prevent conflicting setups.

The main advantage of static identification is that it detects conflicts proactively (i.e. before they actually take place), while dynamic detection relies on monitoring the effects of a "live" conflict. The major disadvantage on the other hand, is that networking experts' knowledge must have been carefully passed to the system through some manual, human-involving way, which is most oftenly error-prone and time-consuming. In this respect, UMF employs knowledge engineering techniques such as ontology, logic programming, and reasoning languages in order to empower static conflict identification and relax the requirements of apriori given knowledge.

One reason for utilizing ontology languages (and in particular the Web Ontology Language - OWL2.0) in static conflict identification (instead of yet another information model) is their flexibility and their rich, yet generic enough, description/representation (in other words "knowledge capturing") capabilities. A UMF knowledge base (KB) is built **offline** and it consists of some formalisms, structure (in the form of graphs) and instances/individuals reflecting the particular UMF instantiation. Notably, the structure is designed once-and-for-all at project design-time, while the task of populating it with individuals is carried out on a per-UMF-instance basis by the network experts in charge.

The second, and probably most important reason for choosing a Resource Description Framework (RDF) -based ontology language such as OWL is the potentials it exhibits for further reasoning and integration with rules (e.g. defined using the Semantic Web Rule Language - SWRL). The main idea is to capture the semantics of parameters, values, operators, actors (e.g. NEMs), and goals/targets during the project design-time, so that:

- the network expert is provided with the tools to express his/her (**sufficient) knowledge** when instantiating a UMF system;
- reasoners (or other mechanisms) can (a) infer **additional knowledge** based on those semantics, and (b) detect conflicts at **runtime** by monitoring state/configuration changes.

"Sufficient knowledge" above refers to the set of facts that will enable the reasoning mechanisms both to infer additional knowledge and to detect the conflicts (e.g. the facts that "*parameter A affects parameter B*", and "*parameter B affects parameter C*" constitute sufficient knowledge to infer that "*parameter A affects parameter C*", provided that the semantics of "*affects*" are properly captured during design-time). Such rules, as well as rules expressing conflicts can be formally expressed through the logical clauses of most rule- or logic programming- languages.

Obviously, conflicts occur between "*actors*" on parameters, which, in the case of UMF, are NEMs. A second observation is that a conflict is unlikely to occur between any two NEMs that act upon different parameters or even control the same parameter in a similar and cooperative fashion. Provided with these two statements only, it is already clear that the semantics of the "acting upon" (or the "controlling of") parameters have to be captured along with the semantics of the parameter (as a concept) itself. The first categorization of parameters the UMF makes is into comparable and non-comparable ones, defined as having naturally orderable and non-orderable values respectively. Numerical parameters of course fall into the former category. In the very same way, values of parameters are categorized and used accordingly. Values are also distinguished from value sets, which can again be comparable (e.g. `[0,1] < [3,4]`) or non-comparable (e.g. `{nodeA, nodeB, nodeC} in subnetX, {nodeA, nodeD} in subnetY`).

The next concept to be semantically enriched is that of the "affects" association between parameters. The description of the relations of parameters will allow for inferring complex and potentially indirect coupling between them. Instead of using a predicate (as a verb in natural language usually dictates) to represent the relationship, in this case, a class is chosen so that it can be further linked to more properties regarding the characteristics of the relationship. That class, namely "Impact", contains individuals that are used to describe (direct or indirect) coupling of parameters. Linear dependencies can be described through a data property assignment to such individuals, namely "hasStrength". The value of the "hasStrength" property in this case represents the differential of one parameter as a linear function of the other ($Strength = \frac{dy}{dx} = c$, with c being a constant, since y is a linear function of x) and represents the ratio of change in y over a change in x. This kind of linear dependency assumption, although naive and unrealistic for the most part of the networking world, allows the reasoner to infer indirect coupling of parameters previously unknown to the system using the chain rule:

$$if\ \frac{dx_1}{dx_2}=c_1,\frac{dx_2}{dx_3}=c_2....,\frac{dx_i}{dx_{i+1}}=c_i, \qquad (0.1)$$

$$then\frac{dx_n}{dx_{m+1}}=\prod_{k=n}^{m}c_k,$$

$$where\ \ n<m,\ \ n\geq 1\ \ and\ \ m\leq i$$

The consequent of this rule can be easily inferred and calculated by a reasoner, and is what we refer to *additional knowledge*, since it was never manually or automatically injected into the system by the network engineer.

The next question to be asked is "How does all this contribute in defining and identifying conflicts?". The missing piece to answer this question is the actual "intentions" of the actors over the parameters. For instance, - on a still primitive assumption - a NEM might be *minimizing* or *maximizing* a comparable parameter. It might also be *maintaining a value* or *a value set* on a comparable or non-comparable parameter. "minimizes", "maximizes", and "maintainsValueSet" are the basic predicates identified so far to describe the target behaviour of a NEM regarding a particular parameter; meaning they can be applied from the subject/NEM to the object/parameter. Let us denote with *Ax* the behaviour of NEM_A over the parameter *x*, where:

$$Ax=\begin{cases}-1,\ \text{for "minimizes"}\\ 0,\ \text{for "maintainsValueSet"}\\ 1,\ \text{for "maximizes"}\end{cases}$$

Then, the conflict definitions identified so far can be summarized in (organized into 3 conflict types):

a) **DirectOnMinMax**: when a NEM_A is minimizing and another NEM_B is maximizing (or vice versa) the same comparable parameter ($Ax\times Bx<0$)

b) **IndirectOnMinMax**: when a NEM_A is minimizing, and another NEM_B is maximizing (or vice versa) two indirectly coupled parameters *x* and *y*, with $\frac{dy}{dx}=c$, such that $Ax\times Bx\times c<0$

c) **DirectOnDisjointSets**: when a NEM is maintaining a value (resp. a value set) and another NEM is maintaining another value (resp. a value set) on the same parameter, and the values are different (resp. disjoint)

Dynamic conflict identification

As aforementioned, the role of the dynamic conflict identification method is to “catch” during the actual runtime, conflicts and problematic situations that may have been missed during the static conflict identification phase. To do so the dynamic conflict identification method needs to be regularly collecting through monitoring, information related to the derivatives of Key Performance Indicators (KPIs) as a function of the NEM control parameters.

As such the NEMs need to be regularly triggered to report the values of the KPIs of interest as well as the values of their control parameters so that COORD can calculate the matrix A with elements Aij, where Aij is the derivative of KPI i as a function of NEM control parameter j. This means that the derivatives of all KPIs against all NEM control parameters must be calculated. Once this is done the eigenvalues of matrix A are calculated. If all eigenvalues are negative then the set of NEMs can be considered as stable; otherwise there is risk of instabilities and as such the “Identify NEM Coordination Needs” operation may need to be re-invoked to produce an updated list of conflicts which will then be assigned to the available coordination mechanisms.

Therefore during the initialization phase, sets of NEMs are assigned to instances of the dynamic conflict identification mechanisms. The selection and grouping of NEMs under each instance can depend on factors such as proximity of NEMs’ controlled network elements and can also be affected by the outcome of the static conflict identification phase. For example if the static conflict identification mechanism is not sure about

whether the coexistence of a set of NEMs can lead to conflicts, it may allow NEMs to operate in standalone mode (i.e. not grouped under a conflict managing mechanism) and also possibly constrained and assign them to an instance of the dynamic conflict identification mechanism to observe their actual runtime behaviour.

During runtime, KPI and NEM control parameter values are reported and based on the A matrix eigenvalues, COORD can deduce whether NEMs can keep operating in their current mode (and even relax gradually their constraints -if set) or a new grouping under the available conflict managing mechanisms is needed.

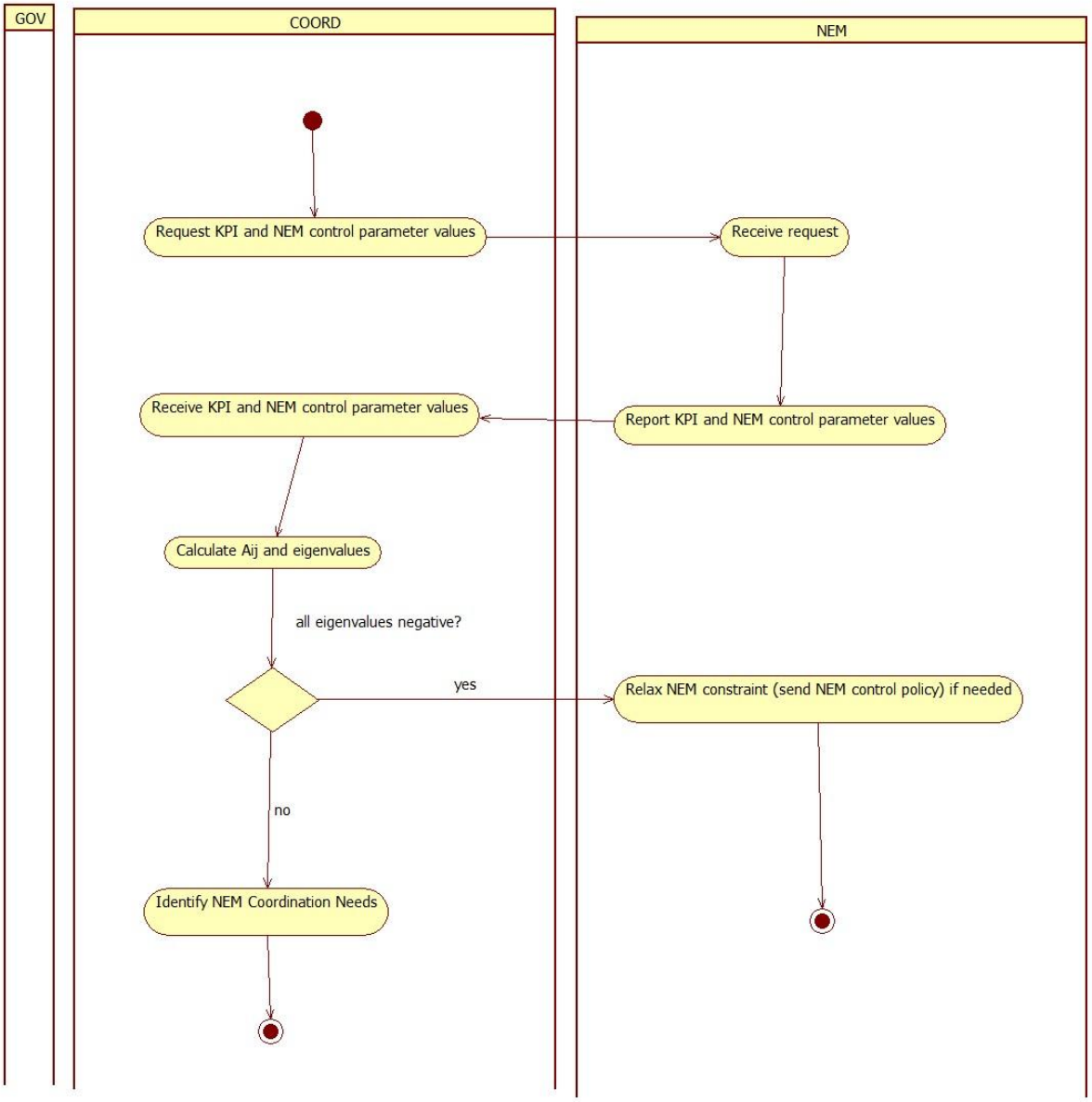


Figure 60. Operation of the dynamic conflict identification scheme during runtime.

4.3.2 Conflict Managing Mechanisms

The following categories of core mechanisms for managing conflicts and concurrent operations of NEMs have been defined: (a) hierarchical optimization, (b) separation in time strategies, (c) centralized multi-objective optimization, and (d) synchronous control theory (and its asynchronous generalization).

Hierarchical optimization deals with NEMs that operate at different time scales; that is they optimize some network parameters with different frequency. The idea here is to set “slower” NEMs as “leaders” and the “faster” NEMs as “followers” and have NEMs optimize their parameters independently and the faster NEMs

seeing the configurations enforced by the slower NEMs as “semi-static”. All NEMs in this approach keep their individual objectives intact and they can operate in parallel at different time scales though.

Separation in time strategies on purpose tries to have only one NEM enforcing a network parameter change at a time. When the NEMs have similar time scales, the simplest approach is to select a NEM randomly. A more optimized way is to select the NEM that leads to the highest global performance (utility). This means that NEMs must be able to predict the effect of their actions and that the notion of utility must be such that all NEMs do understand how their actions can affect it through their parameter changes. When NEMs have different time scales, one approach is to let the NEMs that optimize rather infrequently (or take longer to converge) to converge first and then allow the other NEMs operate till convergence subject to the constraints set by the first type of NEMs. This approach does require though that NEMs must be able to converge to some fixed parameter values -that the other NEMs ideally should be able to keep intact and still operate adequately towards their own objectives- and in reasonable time, so all NEMs get the chance to operate.

Centralized multi-objective optimization deals with NEMs that operate at the same time scale and tries to optimize a global/weighted utility function which combines the utility functions of all considered NEMs. One strong requirement in this case is that all NEMs must be using some explicit utility/objective function for optimization purposes and that their combination is a feasible task. The task of global utility function optimization here is delegated to a centralized entity; it can also be one NEM that acts as such entity with the other NEMs delegating to it the task of finding parameter values on their behalf. The latter case is possible if all NEMs manage the same network parameters and rely on the same monitoring information, as such one NEM alone can set them to values “on behalf” of the other NEMs. In all cases though, there can be only one decision making entity.

Synchronous control theory approaches deal with NEMs (processes/functionalities in general) that operate at the same time scale and are synchronized. The idea here is to have NEMs jointly optimize towards their individual objectives, but taking into account the influence of the other NEMs. This can be done by considering a global/aggregated expression of their utilities (optimization objectives) or an expression of the weighted deviation from some pre-defined per-NEM optimization targets. The same approach can also be generalized to be applied for NEMs that are not necessarily synchronized in time but have the same average frequency of parameter changes. Synchronous control theory differs from centralized multi-objective optimization in that, while the NEMs need to consider amended objective functions, these amended objective functions can be treated separately. This is in contrast to centralized multi-objective optimization where the individual objective functions needs to be considered simultaneously. As such, synchronous control theory approaches can be also solved using a centralized optimizer but also allow for distributed implementation where after the definition of the amended objective functions, the task of their optimization can be delegated to individual NEMs.

Random token

This method for managing conflicts belongs to the separation in time strategies and is the simplest possible mechanism and sets the baseline for all other conflict managing mechanisms. It can viewed as the last resort when other mechanisms cannot be applied, due to NEMs not being able to provide the needed information required by the other mechanisms.

Such (absent) information can relate to:

- Periodicity; i.e. the NEM may not be able to provide it’s time-scale because it is not periodic in its behaviour but rather triggered by events that can only be defined during runtime (e.g. triggered by link utilization thresholds)
- Convergence time; i.e. the NEM cannot provide even an indication of how much time (how many steps) it may require to converge. As such selecting a NEM and allowing it to run until convergence may deprive other NEMs for running for a very long time.
- Performance quantification; i.e. the NEM’s overall contribution to network-wide performance cannot be quantified. As such there cannot be a strict decision if a NEM’s actions are more or less beneficial than other NEMs’ actions. There can be however some general guidelines about the relative importance of a NEM.

Constraints

In such cases, NEMs - identified as possibly conflicting though exhibiting the above characteristics - are grouped under an instantiation of a random token mechanism. This mechanism does not require runtime information from the network but only needs pushing the token probabilistically to the controlled NEMs. The periodicity of the token passing mechanism can be set comparable to the periodicities –if known- of the controlled NEMs.

Initialization

During the initialization phase of this mechanism, i.e. when a set of NEMs are selected to be handled by it, COORD will send the appropriate RegimePolicy to the NEMs instructing them that they should follow the following behavioural pattern “only run one cycle of your MAPE loop when you receive the token and if you need to do so when you are granted the token”. The random token mechanism does not need to establish any channels with KNOW or perform any other interactions during the initialization phase.

Dynamic functioning

During runtime, NEMs once receiving the token they will check if they need actually the token (i.e. they would require to run their optimization cycle, either based on their periodicity or their triggering events)

If the token is needed, the NEM is allowed to run its optimization cycle only once and once finished it returns it to COORD returning to inactive mode. If the token is not needed then the NEM directly returns it to COORD remaining inactive. The above mentioned behaviour is illustrated below

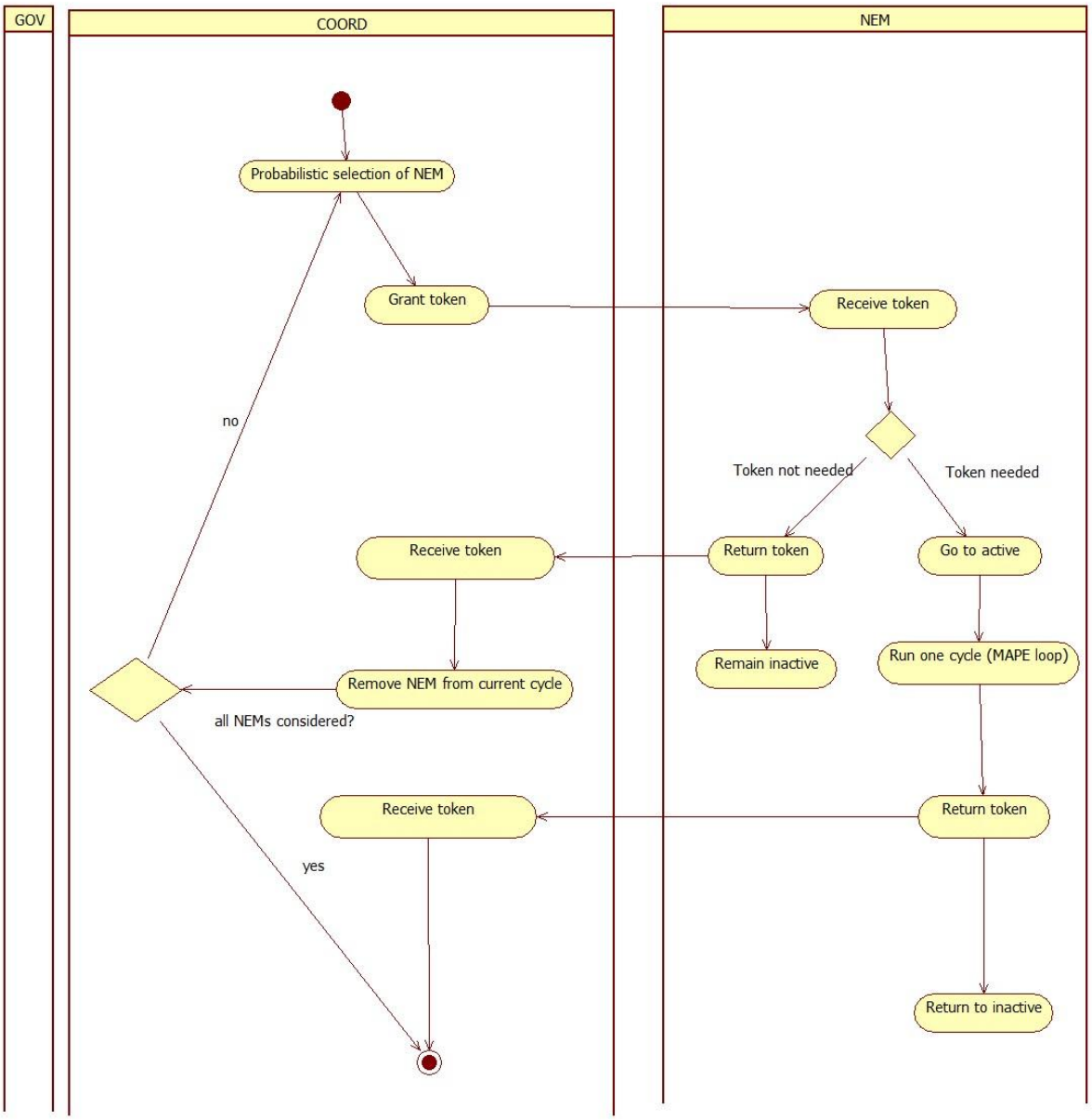


Figure 61. Dynamic functioning of the random token mechanism.

SOUP (Self-orchestration through Utility Predicates)

This method for conflict avoidance can be regarded as an extension of the random token mechanism, with the key differences that in order to be applicable

- NEMs must be able to predict the outcome of their actions (predicted utility)
- NEMs must have the same periodicity, or at least periodicity equal to multiples of the same base periodicity
- NEMs -ideally- should be able to tolerate the temporary suspension of their actions and still converge or should exhibit ”1-step” convergence behaviour; that is they exhibit memory less behaviour.

Constraints

In such cases, NEMs that are identified as possibly conflicting but exhibit the above characteristics are grouped under an instantiation of a SOUP mechanism. The SOUP mechanism does not require any runtime information from the network but only needs during runtime to request and receive the predicted utilities from the controlled NEMs. NEMs report their predicted utilities only when their triggering conditions are met. Then the NEM with highest predicted utility is selected to complete its MAPE loop IF the triggering conditions are still valid. If not then the NEM with the second highest predicted utility is selected etc.

Initialization

During the initialization phase of this mechanism, i.e. when a set of NEMs are selected to be handled by it, COORD will send the appropriate RegimePolicy to the NEMs instructing them that they should follow the following behavioural pattern “report utility when asked and need to be activated, only run one cycle of your MAPE loop when you receive the token and if you need to do so when you are granted the token”. In addition the notion of utility is set to the NEMs during the initialization phase since it needs to be unified among all controlled NEMs.

Dynamic functioning

During runtime, COORD will be requesting from NEMs their predicted utilities with frequency defined by the base periodicity. NEMs which meet their triggering conditions report the corresponding values and COORD will choose the NEM with the highest predicted utility to be activated. If the selected NEM still meets its triggering conditions it will run one cycle (more precisely it will directly enforce its previously selected action); otherwise the NEM with the second highest predicted utility will be selected and so on.

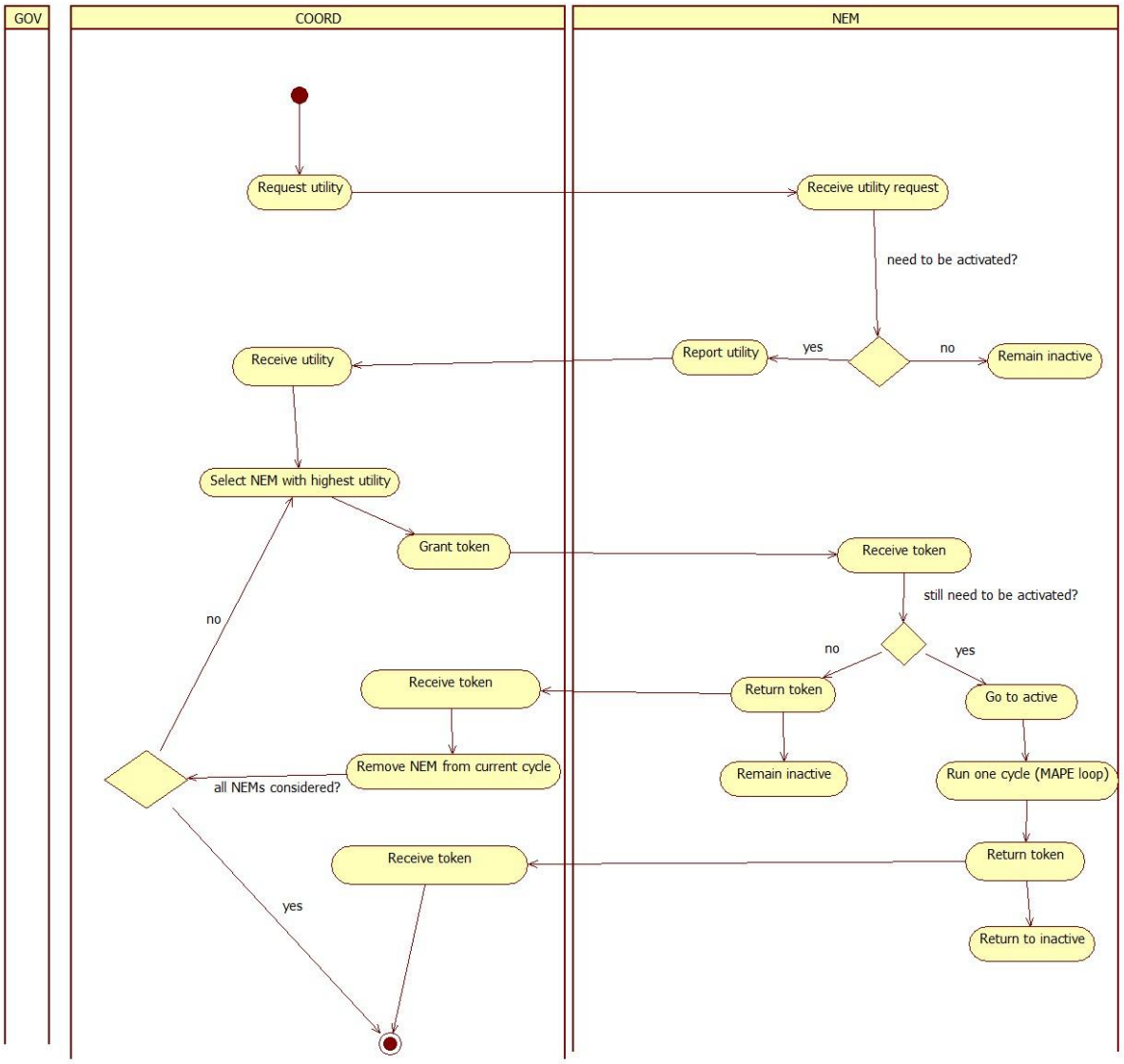


Figure 62. Dynamic functioning of the SOUP mechanism

Centralized multi-objective optimization

As the name identifies, i.e. joint optimization, this section mainly regards the coordination of optimizing NEMs. Therefore, when there is a need for coordination among several optimizing NEMs, a straightforward solution 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 optimization.

Therefore, one first idea is to formulate the NEM coordination as a multi-objective optimization (MO) problem. The general MO problem requiring the optimization of *N* objectives may be formulated as follows:

$$\min \vec{F}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_N(\vec{x})]^T$$

$$\text{subject to } g_m(\vec{x}) \leq 0, m = 1,2,\ldots M$$

$$\text{where } \vec{x} = [x_1, x_2, \ldots, x_p]^T \in \Omega$$

where $\vec{F}$ represents the objective vector, g denotes the constraints and $\vec{x}$ is a *P*-dimensional vector representing the decision variables within a parameter space Ω. The space spanned by the objective vectors is called the objective space, while the subspace of the objective vectors that satisfies the constraints is called the feasible space. Note that because $\vec{F}$ is a vector, if any of the components of $\vec{F}$ are competing, there is no unique solution to this problem. Instead, the concept of non-inferiority (also called Pareto optimality) must be used to characterize the objectives. A non-inferior solution is one in which an improvement in one objective requires a degradation of another. Multi-objective optimization is, therefore, concerned with the generation and selection of non-inferior solution points, which are also called *Pareto optima*.

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. These methods produce as outcome a single solution rather than a Pareto Optima.

Constraints

The NEMs should be able to provide their analytical function of the NEM's utility through the NEM Instance Description, to allow disabling their work and to receive a NEM Control Policy that would imply for the NEM a Set parameter value action only.

Initialization

At the initialization phase, COORD constructs the optimization objective and method that will be used to jointly optimize for the set of NEMs under the control of each instance of the centralized multi-objective optimization mechanism. COORD also sends to the NEMs the appropriate RegimePolicy that instructs NEMs to disable their working (only perform monitoring), report the monitored information when requested and simply enforce the network parameter value whenever they receive it from the centralized optimizer.

Dynamic functioning

During runtime, upon request from COORD (dictated by the periodicity of the centralized optimizer), NEMs report to COORD the monitored information they would be using to solve their individual optimization problems. Having received this information COORD proceeds to solve the multi-objective optimization problem and calculate the network parameter values that NEMs will need to enforce to the controlled network elements. These values are sent to the corresponding NEMs and are enforced and remain valid until the next cycle of the centralized optimizer.

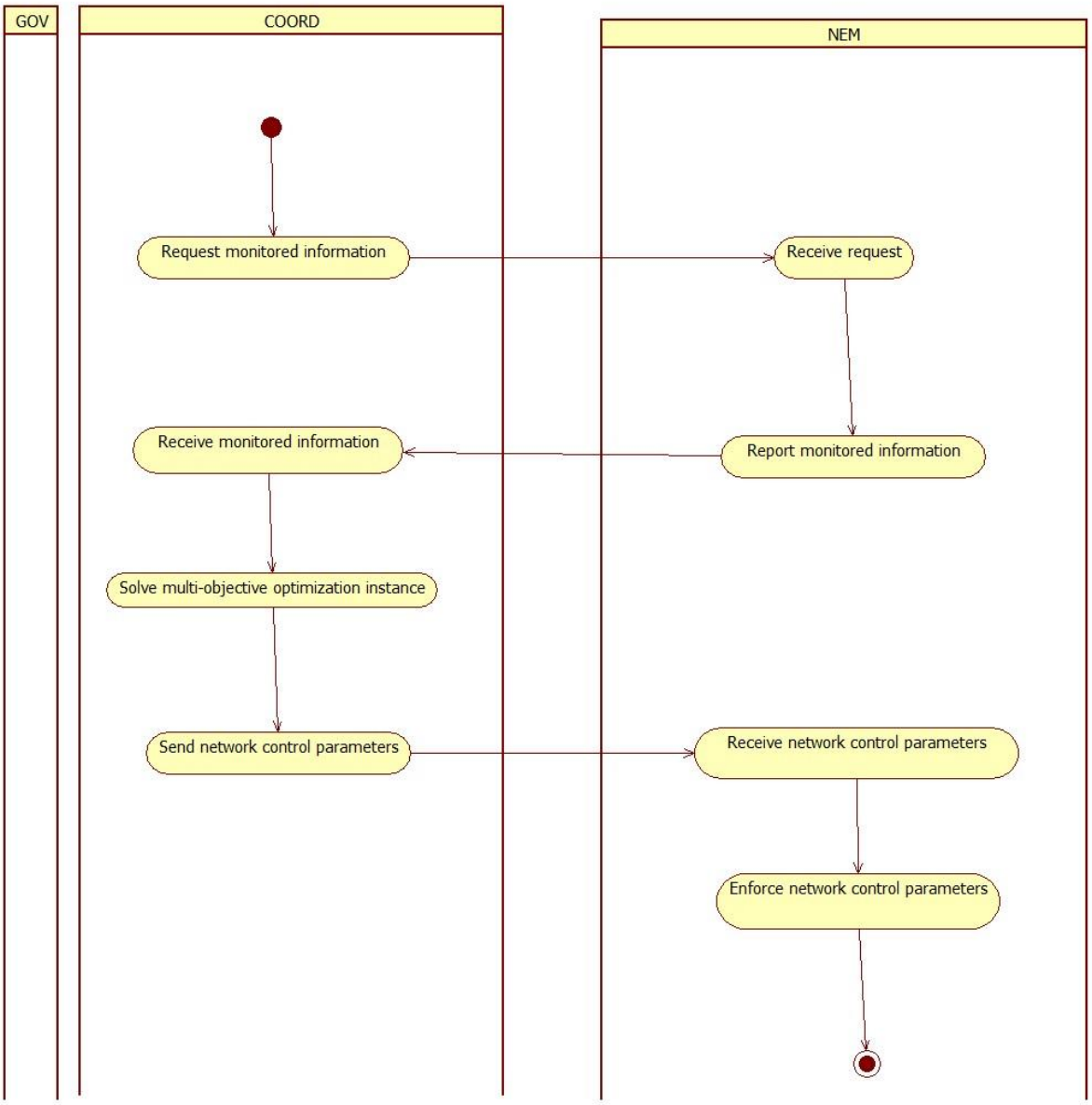


Figure 63. Dynamic behaviour of the centralized multi-objective optimization mechanism.

Hierarchical optimization

This method for managing conflicts is based on the fact that control loops (NEMs) in a communication network may operate on different time scales, according to the dynamics of the resources which are controlled. By enforcing different time scales of two or more NEMs, one defines a hierarchical control system which is denoted here as hierarchical optimization. After each control action (e.g. parameter modification) of the slow NEM, the fast NEM acts rapidly and converges. Hence the slow NEM is seen as quasi-static to the fast NEM. If we denote the periodicity of the fast NEM as T_f and that of the slow NEM as T_s, then convergence of the hierarchical system is guaranteed if $T_f / T_s \to 0$, or in practice, a small value. In Game Theory, the slow NEM can be considered as a "leader", while the fast NEM is considered as a "follower"; and the optimal point of operation is known as Stackelberg equilibrium.

Constraints

For this method to be applicable, the NEMs must be able to operate periodically. This mechanism does not require any runtime information from the network or from NEMs themselves.

Initialization

During the initialization phase, COORD simply sends to the NEMs the periodicities they should follow (RegimePolicy). These periodicities can be the default NEM periodicities but also adjusted periodicities if needed to "construct" the hierarchical optimization system and these adjusted periodicities can be tolerated by the NEMs.

After this, the NEMs in principle are allowed to operate otherwise unconstrained.

Dynamic functioning

This conflict managing mechanism does not need any interactions with NEMs or any other core block during runtime. The NEMs controlled by it may be part of an instance of the dynamic conflict identification mechanism; as long though as there are no alarms raised by the dynamic conflict identification mechanism, the NEMs can keep operating as instructed during the initialization phase.

Synchronous control theory

This method for managing conflicts differs from the multi-objective optimization method in that, while NEMs controlled by it, take into account the objectives of other NEMs, each NEM can still on its own calculate and enforce its actions without relying in a centralized optimizer for doing so. However, since this distributed approach may raise scalability concerns due to the increased communication overhead needed if direct NEM communication is allowed, it may be preferred to employ a centralized multi-objective optimization approach where NEMs report to COORD the required information during runtime, COORD performs the necessary calculations, and NEMs simply enforce the outcome of these calculations

Constraints

For this method to be applicable, NEMs must be able to operate at the same time scale, synchronously, or -under certain conditions- asynchronously but with the same average frequency. In addition NEMs must be able to provide their analytical function of the NEM's utility through the NEM Instance Description, to allow disabling their work and to receive a NEM Control Policy that would imply for the NEM a Set parameter value action only. This method requires from NEMs to be able to state their optimization targets upfront and also to be able during runtime to report to COORD their KPI values. COORD, once receiving this information from all NEMs, it can calculate the instantaneous value of the objective function for each NEM as well as the value of the network parameters they need to enforce.

This method can be also combined with hierarchical optimization, with groups of NEMs with the same/similar time scales being considered under instances of the synchronous control theory method (one instance per group) and with hierarchical optimization being applied at the group level.

Initialization

At the initialization phase, COORD constructs the optimization objective for each NEM. COORD also sends to the NEMs the appropriate RegimePolicy that instructs NEMs to disable their working (only perform monitoring of their KPIs), report the monitored information when requested and simply enforce the network parameter value whenever they receive it from the centralized optimizer.

Dynamic functioning

The dynamic behaviour is similar to the dynamic behaviour of the centralized multi-objective optimization; the only difference being in the way the centralized optimizer solves the multi-objective optimization problem internally. Rather than solving one problem to calculate the network parameters to be enforced, multiple separate problems are solved in parallel, each of them leading to the calculation of the network parameters that need to be enforced by each NEM.

Multi-priority event driven separation in time

The main idea is based on avoiding possible conflicting behaviour of NEMs issues at design time by setting up appropriate **operation requirements** and validate that NEMs operation is not violating these requirements.

Some NEMs are operating asynchronously and are executed on an event-driven basis. Some NEMs are considered to be "always ON": their objective is not to fix or modify a network parameter and then stop (e.g. some act as knowledge production mechanisms and perform continuous monitoring and spontaneous knowledge report, on-request or on-event). Some NEMs complete their lifecycle by fixing or/and modifying parameter values but assumed to remain ON so that they can react in case of deviations from their optimization targets. Therefore setting up a multi-priority, event-driven separation in time mechanism for their operation could be an appropriate way for avoiding conflicts.

NEMs performing in different context could run in parallel without conflict. So, as a first step, the mechanism needs to classify NEMs according to the domain of their operation (access, core, etc) the SON type they address (i.e. self-healing, self- configuration and self-optimisation) as well as the SON use case they target on (CCO, ICIC, ES etc) within the SON types. NEM's operational state can be further detailed using the MAPE loop: NEM needs to collect information from various sources in the network, analyze them, reach a decision about actions that need to be executed and finally to be informed about the results of these actions. Based on these considerations the MAPE phases are considered regarding at which phase a NEM can be interrupted, if this is needed to avoid conflicts, and allow another NEM to execute a task of a higher priority. These are defined at design time and followed during run time.

MAPE loop has been chosen as providing a well-known lifecycle and capturing the operational states of the NEMs. MAPE loop incorporates four (4) phases and NEMs operation can be linked to those MAPE phases; therefore, the transitions are easier to identify and manage. The COORD block will have the responsibility to coordinate these interrupts by notifying the appropriate NEMs to pause and reset/resume as appropriate. Furthermore, the COORD will need to know beforehand whether an action in one domain (e.g., access domain) will affect other domains (e.g., core) so as to synchronize the operation of NEMs operating in other parts of the network.

The approach is based on each NEM context for identifying "possible" conflicts and on the SON features for identifying ways to avoid conflicts. The mechanism is addressing the "operational" NEM state; key assumptions and features are presented in the following list:

- Conflicts
    - NEM context has been defined to include network domains, targeted elements as well as input/output parameters,
    - Possible conflicts can be identified whenever a new NEM is registered or updated; check is performed regarding the NEM's context,
- Conflict avoidance requirements
    - NEMs have been analysed and assigned a specific SON feature; so each NEM can be self-healing **XOR** self-optimisation **XOR** self-configuration,
    - A self-healing NEM can run always,
    - A self-configuration NEM can run if self-healing NEM is not running,
    - A self-optimisation NEM can run if self-healing NEM **AND** self-configuration NEMs are not running.
- Conflict avoidance (NEM Control Policy **typeof** Regime Policy)
    - A self-optimisation NEM shall interact with COORD during operational state when implementing MAPE loop – at any MAPE phase transition: If a self-healing OR self-configuration NEM needs to run, self-optimisation NEM shall be halted,
    - A self-configuration NEM shall interact with COORD during operational state when implementing MAPE loop – at any MAPE phase transition: If a self –healing NEM needs to run, self-configuration NEM shall be halted,
    - A self-healing NEM shall interact with COORD in order to ensure all involved NEMs are halted.
- Resuming/start over
    - **The Monitor phase can be interrupted**
        - Considerations are required for time-critical data,
        - Monitored data can be discarded; therefore monitoring phase need to start over,

        - Monitored data can be kept until time out, if still valid, NEM can proceed to the Analyse phase.
    - **The Analyse phase can be interrupted**
        - If sufficient data have been gathered, the NEM can proceed to analysis during interruption,
        - Time criticality is an important issue and analysis should be performed in valid data,
        - If the gathered data is not sufficient the operation may continue after resuming.
    - **The Plan phase could be interrupted**
        - Special attention to avoid ping-pong or oscillations effects,
        - Consider the affected resources by each action
    - **Execution phase**
        - If the execution is distribution of information/knowledge it can be interrupted,
        - The knowledge could be obsolete after resuming, a time-tag might be needed,
        - If the execution involves configuration actions (e.g. parameter set) the corresponding parameters should be checked.
            - In case of conflict, the action should be postponed and the process should start over,
            - In case of no conflict the action can be performed after resume.

Constraints

Each NEM should provide information on

- The SON feature it is addressing (self-healing, self-configuration, self-optimisation),
- The network domain it is operating in (access, core, etc),
- The network element it is to be deployed,
- The input parameters,
- The parameters to impact (output, set).

Most of these items have been included within the NEM manifest and instance description.

Each NEM should be able to halt and then figure out whether to resume or start over. This could be identified by COORD using global knowledge (on time critical measurements, analysis, change in configurations etc.).

Initialization

During the initialization phase of this mechanism, i.e. when a new NEMs is deployed or updated, COORD will send the appropriate RegimePolicy to the NEMs instructing them that they should follow the following behavioural pattern “interact with COORD before proceeding to any of the MAPE phase”.

Dynamic functioning

The dynamic behaviour of this mechanism is illustrated through the following workflow

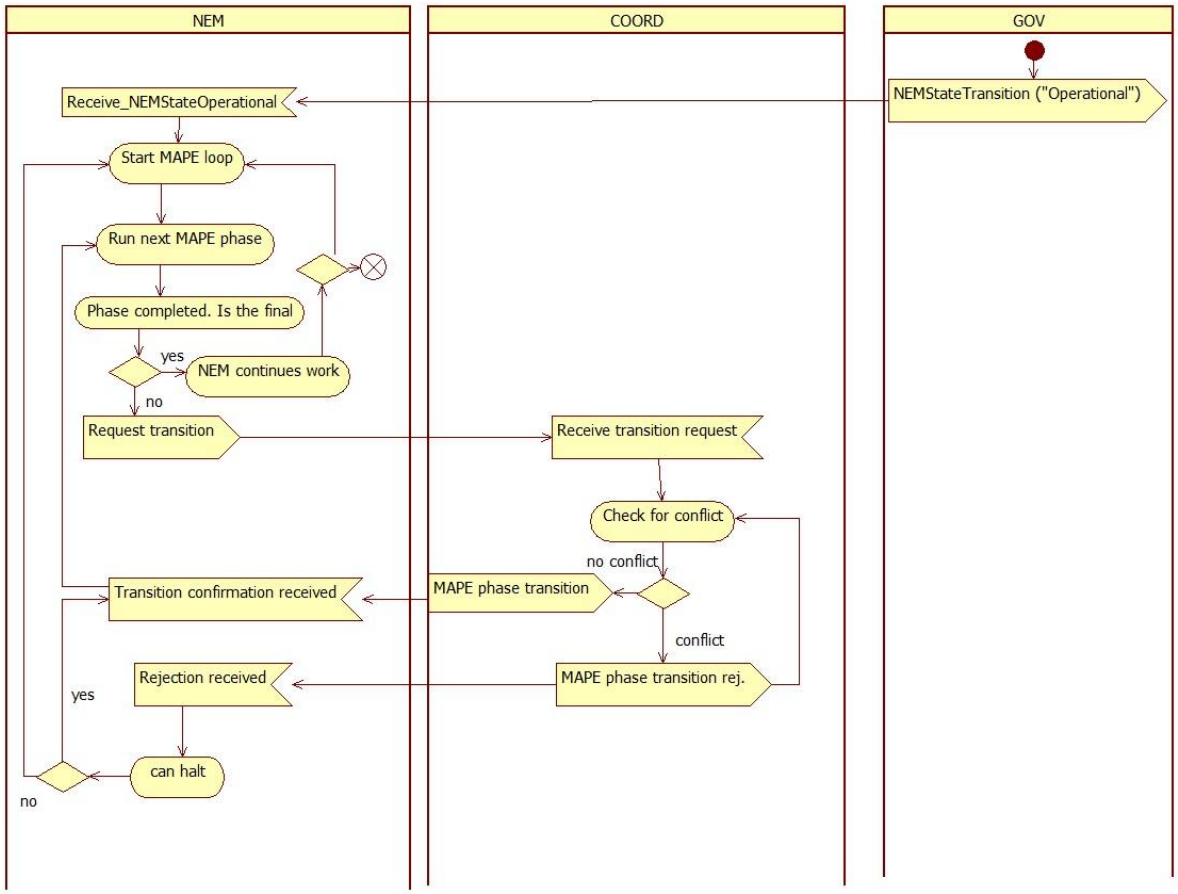


Figure 64: Dynamic operation of the multi-priority event-driven separation in time.

The above presented workflow can be traced as follows:

- Once NEM is receiving the transition to operational state, MAPE loop is initiated :
- Each time the NEM is completing a MAPE loop phase it is checking whether it has completed the last phase (i.e. execution)
    - If yes, the NEM is checking whether its operation is continuous or on demand
        - If on demand, NEM operational state is over,
        - If continuous, NEM is starting over (transition to Start MAPE phase process)
    - If no, the NEM is checking with COORD whether it can move to next MAPE phase
        - COORD is cheeking for any conflict based on NEM information
        - If no conflict, COORD is confirming MAPE phase transition
            - NEM is starting the next MAPE phase (transition to MAPE phase process)
        - If there is potential conflict, NEM is notified to halt, and the COORD will re-check until confirm transition
            - If the NEM can halt it is waiting for MAPE transition confirmation from COORD
            - If the NEM cannot halt, the operational state is starting over.

5. Positioning and Interworking of UMF with Existing and Emerging Network Frameworks

5.1 Positioning and Interworking of UMF with eTOM

For an effective interworking of UMF (functions and a set of data as inputs and outputs) with eTOM (a business process framework that describes all the enterprise processes required by a service provider, and analyses them to different levels of detail according to their significance and priority for the business), we need to first understand the relationships between processes and functions according to [17]

Process is defined as “the flow of activities to solve a particular business problem, or part of it. At early analysis stages for processes, the means of availability and how the data flows are not significant. Whether or not data is handed over or accessed in a central database is not addressed. However, processes are concerned with the triggers that set them into action”.

Function is described as a unit of processing together with its associated data inputs and outputs.

A process will typically make use of activities in a number of functions. Multiple processes may employ a given function. Thus, there is in principle a many-to-many mapping between process and function.

The picture below depicts the application of these definitions to UMF functions and data and eTOM processes:

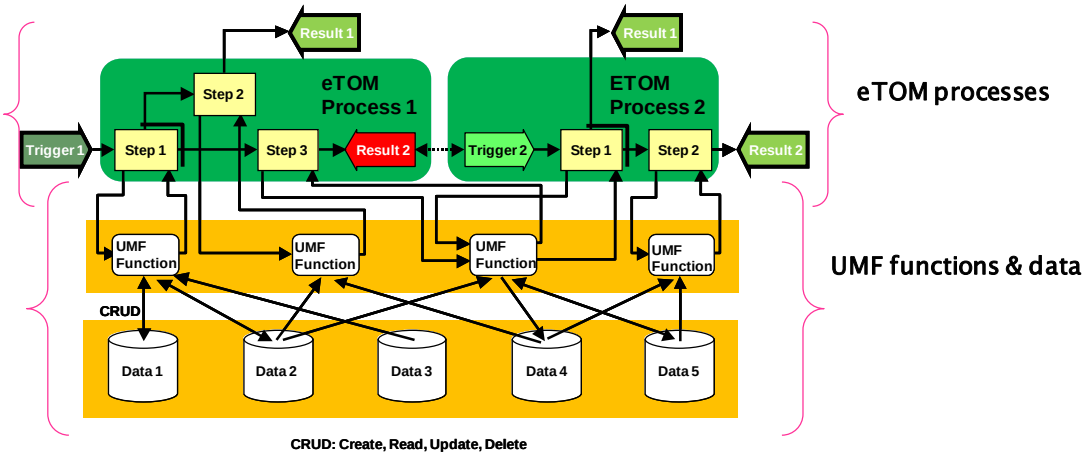


Figure 65: eTOM Processes and UMF functions based on GB921 eTOM

eTOM processes: mainly the UMF functions and data are called by the eTOM operational part.

Furthermore, UMF introduces functions to manage the so-called NEM, while NEM are assigned to Resources to deliver specific services. In this regard, we add the “NEM management & Operations”. This layer gathers specific functions to manage NEMs such us coordination related functions for conflicts management or Knowledge related to NEM information analysis or building.

Other functions are concerned with all eTOM layers including the “NEM management & Operations layers”. The policy derivation function, the knowledge functions about services, resources and NEMs, etc. The picture below illustrates the high level mapping of eTOM to UMF functions.

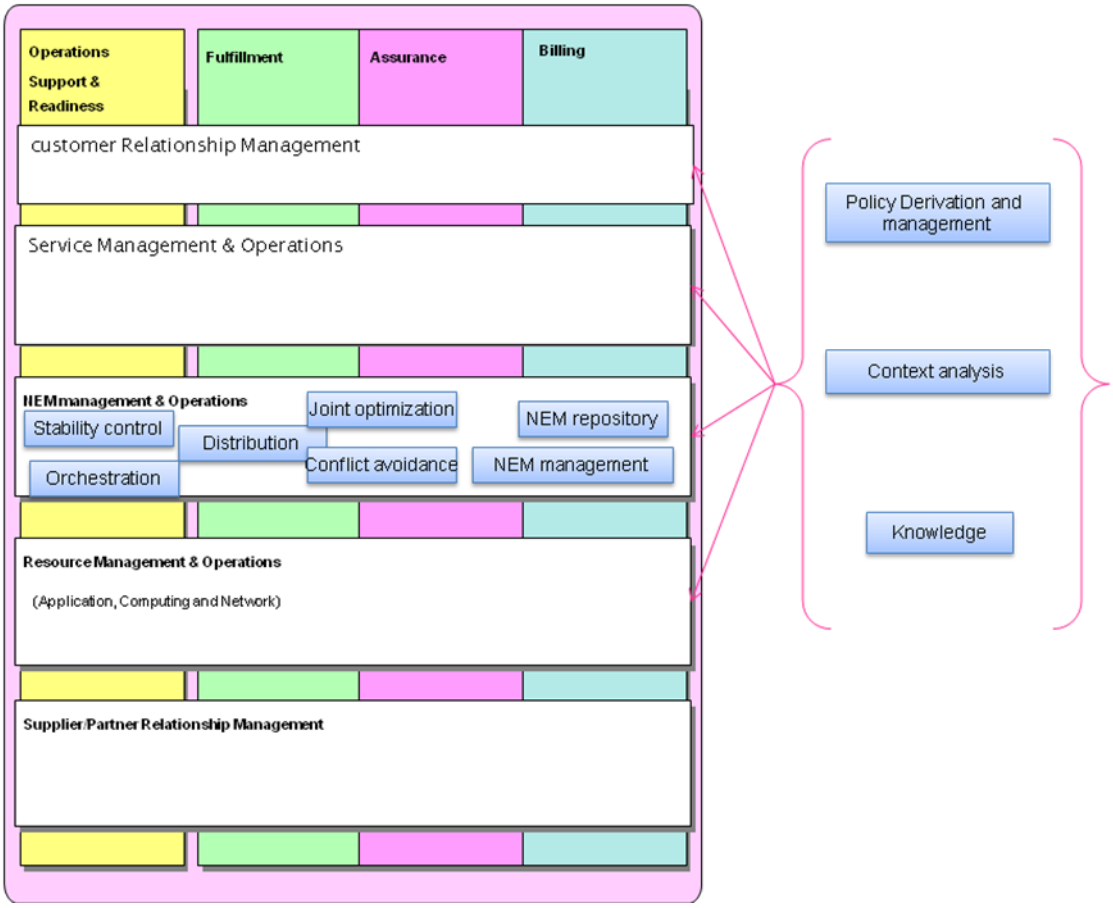


Figure 66: Linking eTOM with UMF functions

The mapping to eTOM requires adding a new layer, called "NEM management & Operations", which embeds some of the functions and operations in Governance, Knowledge and Coordination. NEMs are assigned to Resources layer and deliver specific services.

5.2 Positioning and interworking of UMF with Software Defined Networks

Software defined Networks (SdN) and Network Functions Virtualisation (NFV) enable fully exploitation of service-aware networks, devoid of technology-specific attributes, through provision and management of dynamically network functions and services. UMF core mechanisms and NEMs can form the basis for software-driven networks architecture development. In this context, the UMF interworking with SdN and NFV architectures is presented in this section, indicating that UMF and SdN/NFV can largely benefit from each of other.

Software defined Networks and Network Functions Virtualisation (NFV) are currently under 2nd round of developments. The first round developments were in the form of programmable networks technology [14]. Various solutions are currently proposed by vendors and standardization is still underway. As such the proposed UMF interworking with SdN / NFV context is to be understood as a proposal and not a final view. In the following sections, we highlight the main SdN concepts that are relevant for discussing UMF interworking and we present three interworking scenarios.

5.2.1 SdN main concepts

In this section, we present some of the main concepts that are discussed around Software defined Networks (SdN). The list is not exhaustive as discussions are still underway in several bodies or forums. We can highlight the following initiatives:

- ITU-T SG13: Future networks. An SdN framework is defined and under development.
- Open Networking Foundation (ONF).
- IETF ForCES and IETF SDNRG.

An SdN related standardization activity is ETSI NFV which mainly addresses the visualisation of networking functions. It will be discussed in section 5.2.1.

Figure 67 and **Figure 68** are highlighting the main discussed SdN concepts:

- A decoupling of data plane and control plane with the introduction of a programmable control plane.
- Programmable control plane Southbound Interface (aka Control Data Plane Interface in ONF) towards Data plane: e.g. Openflow, ForCES to control data plane related network devices.
- Northbound Interface relying on APIs between applications and the SdN control plane. APIs between the SdN control and applications layers, enable applications to operate on an abstraction of the network, leveraging network services and capabilities without being tied to the details of their implementation. SdN makes the network not so much “application-aware” as “application-customized” and applications not so much “network-aware” as “network-capability-aware”.
- Southern Interfaces: The OpenFlow / ForCES protocols are deployable between network infrastructure SdN-enabled devices and the SdN control software. OpenFlow uses the concept of flows to identify network traffic based on pre-defined match rules that can be statically or dynamically programmed by the SdN control software. It also allows configuration on how traffic should flow through network devices based on parameters such as usage patterns, applications, and cloud resources. ForCES protocols are used for communications between Control Elements (CEs) and Forwarding Elements (FEs) in a ForCES enabled Network Element (ForCES NE) - RFC3654
- East-West Interface between Control plane/network domains or controllers.
- System Management concepts are not yet explicitly and uniformly integrated in SdN/NFV, however further developments on this subject are expected in the next period.

Some differences exist between the ITU-T, IETF, ETSI and ONF views. For example, it is not clear how “ONF Network Services” that are part of the ONF SdN Control Software is related to ITU-T Network OS, Programmable Control Plane or SdN services/applications.

The frontier between control and management planes is also becoming blurred. Management plane is not explicitly mentioned and is globally missing in the description. It is important for UMF which is dedicated to self-management and self-x functions that are typically targeting management plane but are also blurring the frontier with control plane.

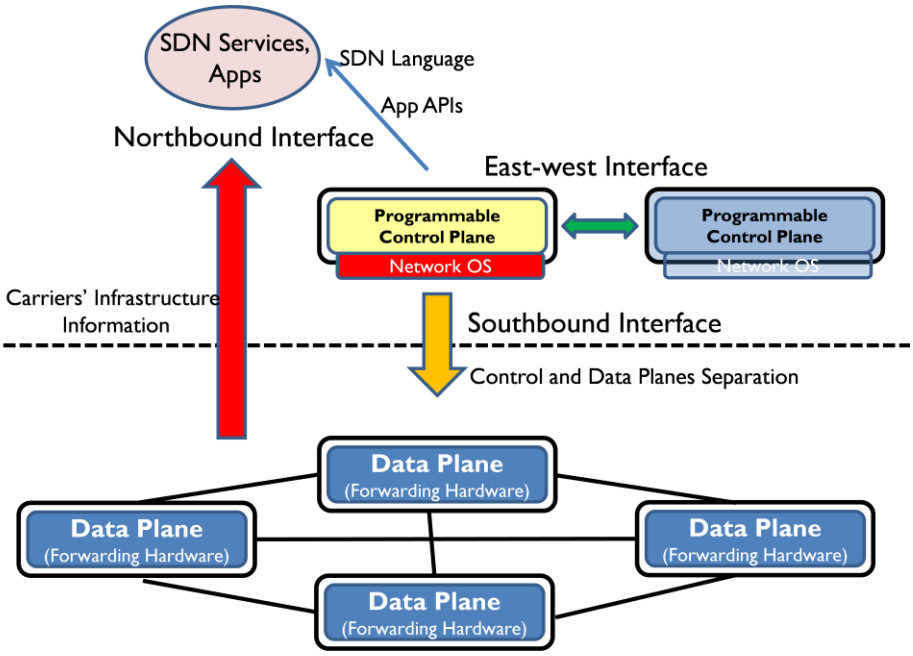


Figure 67: SdN Framework as discussed in ITU-T SG13

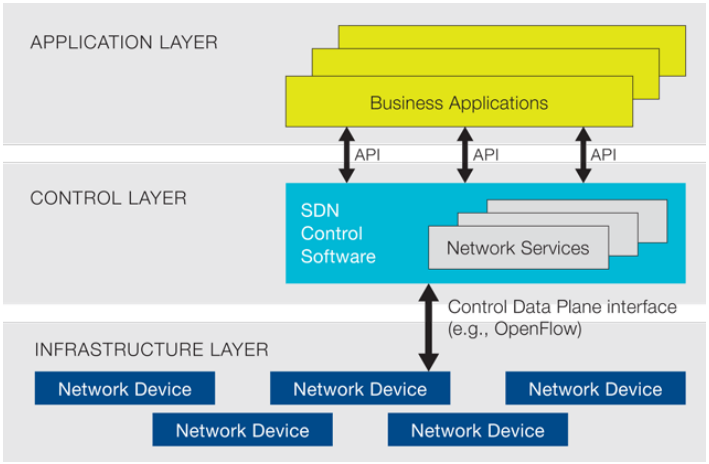


Figure 68: SdN framework as discussed in ONF

5.2.2 Connecting UMF to SdN enabled infrastructures

Figure 69 presents the UMF framework with the three UMF core blocks and the NEMs and related interfaces.

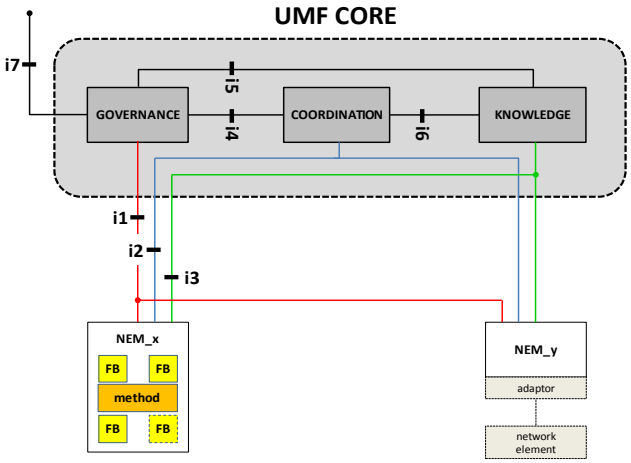


Figure 69: UMF framework

We need to distinguish the UMF Core blocks from the NEMs for this mapping.

UMF core blocks are in charge of the needed functions to manage NEMs and ensure self-management driven by operator goals. This is related to management plane. SdN applications could benefit from UMF core blocks in the case of self-management of SdN applications. In this case, SdN applications would be similar to NEMs.

Moreover NEMs are containing the algorithms/methods part of the self-x functions and as such they contain different and specific protocols for resource configuration of network elements. UMF specifications don't address the NEM adaptation or gatewaying to SdN enabled network element and in particular to the OpenFlow-enabled network device from any vendor, including switches, routers, and virtual switches. This needed adaptation could benefit from the SdN frameworks and could be facilitated by the programmable control plane.

We can also discuss the possible NEM inclusion into the programmable control plane. SdN control plane includes only the functions that can control only SdN-enabled network device from any vendor, including switches, routers, and virtual switches. SdN controllers provide visibility and control over the network, they can ensure that access control, traffic engineering, quality of service, security, and other policies are enforced consistently across the network infrastructures, including branch offices, campuses, and data centers. As stated above, NEMs are management applications which include self-x methods and at least one closed control loop enabling self- functionality of a network element. It acts on the resources which are managed by the NEMs. These two definitions are mainly disjunctive and as such without a revision of the scope of the SdN control plane, the NEMs which are applicable to SdN enabled domains and the NEMs which are applicable to other networking domains cannot be easily fitted in the control plane functionality.

As such we have 3 options:

- Option 1: to not consider as appropriate the inclusion of the NEMs in the current SdN control plane.
- Option 2: to extend the definition of the control plane to include management functionality, including self-management. In this case it makes sense to consider as appropriate the NEMs inclusion in a revised SdN control plane.
- Option 3: to extend the SdN framework to include the management functionality explicitly (i.e. in the current version system management concepts are not yet explicitly and uniformly integrated in SdN/NFV) with control & management plane separation.

In the following sections, we consider only option 3 as a target.

We present three scenarios: the first scenario considers the NEMs as SdN applications. The second scenario considers UMF as a management integration framework for SdN with Non SdN Enabled Infrastructures. The third scenario considers the UMF core positioning in a revised SdN architecture.

Scenario 1: NEMs as a SdN app

In this scenario, we consider that a NEM_for-SdN performs a self-x function on an SdN enabled infrastructure. Such a NEM is the Virtual Infrastructure Management (VIM) (see section 3.3.6). To support this scenario, we mention that ONF is discussing the introduction of some components at the level of a network controller that is related to network services. From the ITU-T perspective, it could be related to the Network OS role.

Figure 70 is illustrating the NEM_for-SdN positioning. First, the related to NEM_for-SdN needs to support the APIs that are provided to SdN applications. NEM_for-SdN needs also to support the UMF interfaces with the three UMF core blocks. In this context, it is difficult to foresee how SdN applications and UMF Core blocks could be connected as they are driven by the same goals. Second, in this context it may be a good idea to bring the UMF interfaces closer to the SdN APIs. Third, the NEM adaptor is relying on the programmable control plane functionality provided by Network OS. As such the developments of the Network OS would need to include the right level of abstraction usable by the NEM_for-SdN.

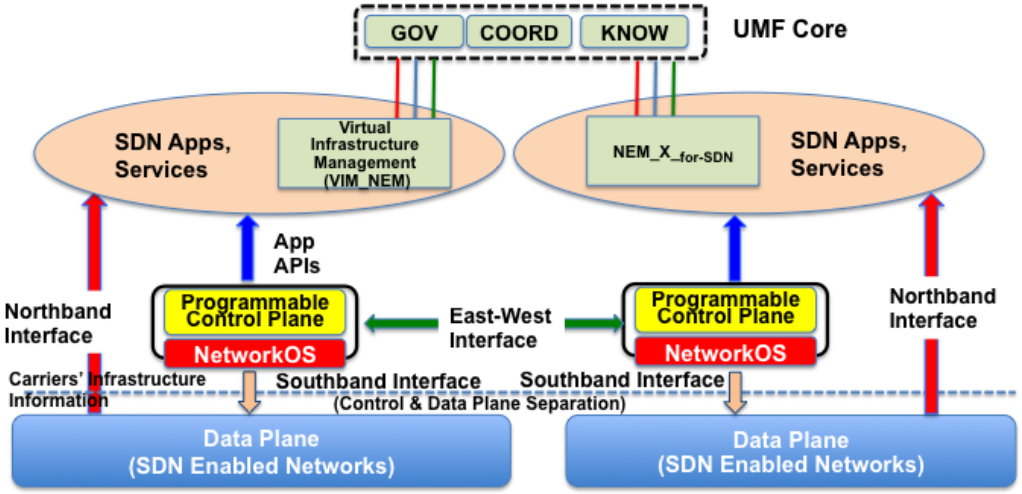


Figure 70: NEMs in SdN enabled Networks as an SdN app

Scenario 2: UMF as management integration framework for the management of SdN with Non SdN Enabled Infrastructures

In this scenario we consider that a NEM_X applicable to a non SdN enabled infrastructure which is not a part of the SdN applications domain as illustrated in **Figure 71**. In this context, the positioning of the UMF core appears to be more obvious than for the previous scenario: UMF interfaces are used to manage NEMs or SdN applications.

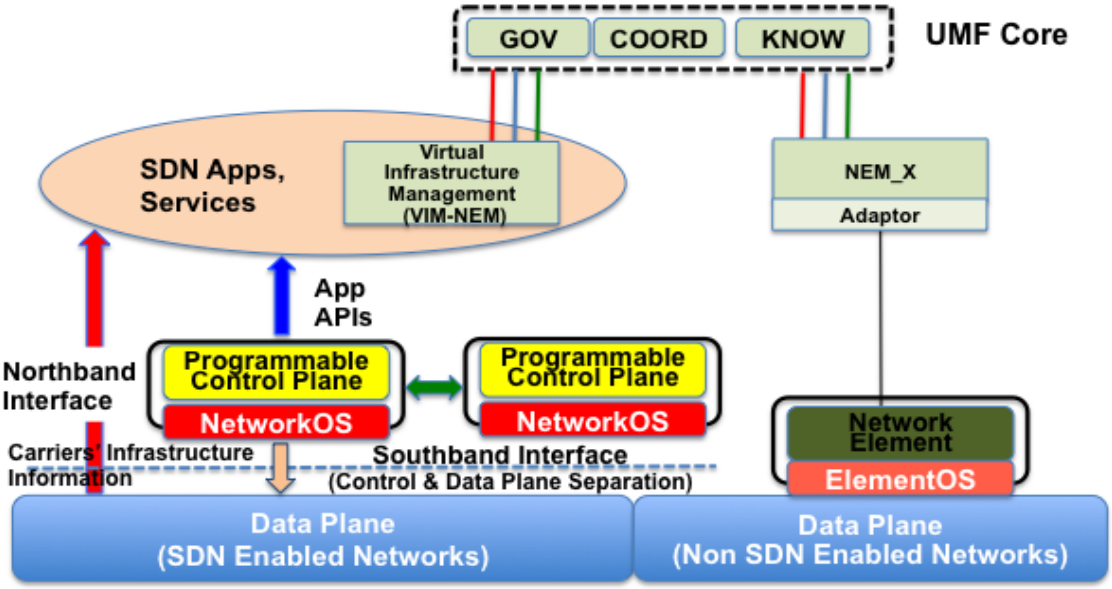


Figure 71: NEM as an SdN app

As such UMF could be established as a management integration framework for the non SdN and SdN enabled infrastructures.

Scenario 3: UMF core positioning in a revised SdN architecture

As presented in Section 5.2.1 System Management concepts are not yet explicitly and uniformly integrated in current established SdN standardisation fora (IETF FORCES & SdNRG, ONF, ITU-T SG13: Future networks/ SdN, ETSI NFV), however further developments on this subject are expected in the next period.

Due to the extreme dynamic nature of the SdN enabled infrastructure it would be appropriate to propose a separation of the control and management planes that could be developed and maintained separately. The control & management planes separation enables the positioning of the full UMF core in the SdN revised architecture which is presented in the next figure.

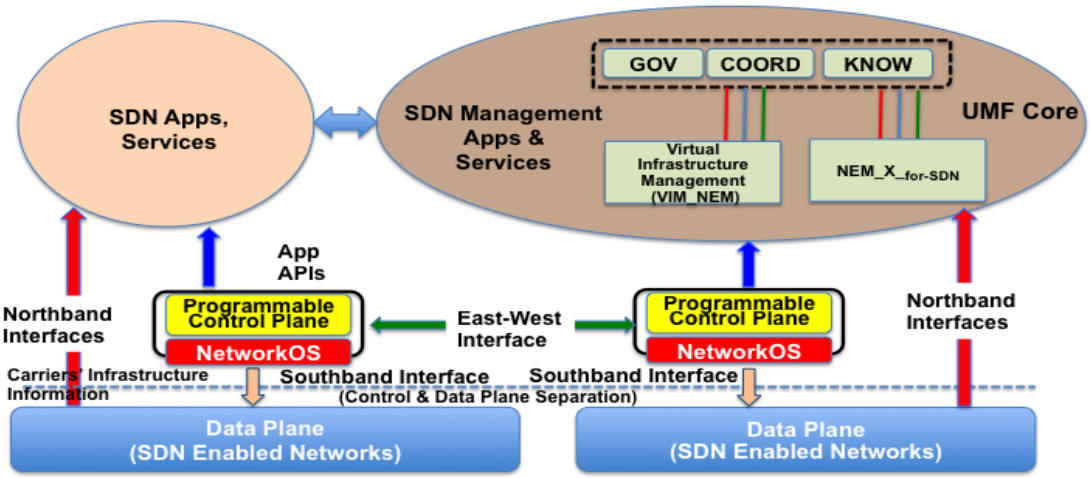


Figure 72: UMF position in a revised SdN framework

One development for a revised SdN framework [15] as presented in the previous figure considers the control and management separation and it also proposes the transition from the current SdN as "application-customized" towards "application-aware" infrastructures and for applications to be "network-aware". It

proposes a service-aware and management-aware network control infrastructure for heterogeneous networks (i.e., wired and wireless) that uses software driven features for the elaboration, development, and validation of networking concepts. It aims also at optimal integration of the connectivity and management layers. It operates across multiple network environments and on top of private and public network clouds utilising fixed and mobile virtual resources, OpenFlow enabled network devices like switches and routers, and networks of Smart Objects. **Figure 73** sketches such SdN revised framework with the introduction of UMF Core blocks and NEMs.

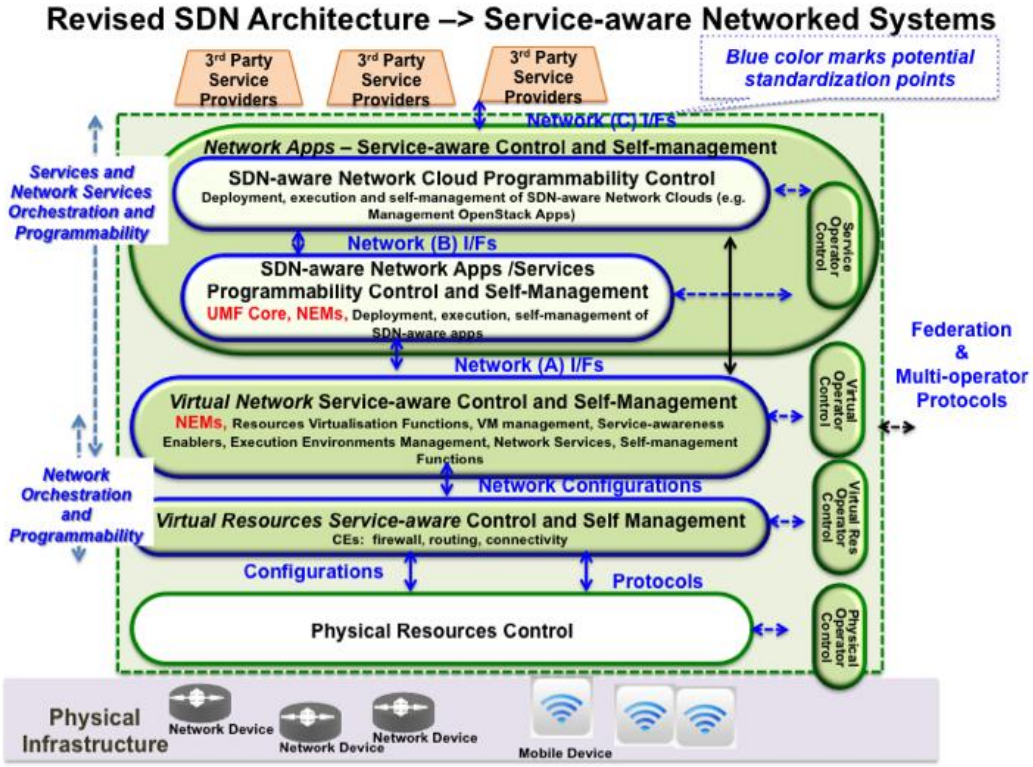


Figure 73: SdN Revised Framework

5.3 Positioning and interworking of UMF with Network Functions Virtualization Architecture

Network Functions Virtualisation aims to address the problems related to maintenance of a large and increasing variety of proprietary hardware appliances which are rapidly reaching end of life, requiring much of the procure design- integrate-deploy cycle to be repeated with little or no additional revenue benefit. These problems are addressed by leveraging standard virtualisation technology to consolidate many network equipment types onto industry standard high volume servers, switches and storage, which could be located in Datacentres, Network Nodes and in the end user premises. It should be noted that an industry specifications group - Network Functions Virtualisation (NFV) [16]- was formed in January 2013 under the European Telecommunications Standards Institute (ETSI). The ETSI NFV work is highly complementary to Software Defined Networking (SdN) as these topics are mutually beneficial but are not dependent on each other. Network Functions can be virtualised and deployed without an SdN being required and vice-versa. The basic NFV architecture is presented in the following figure.

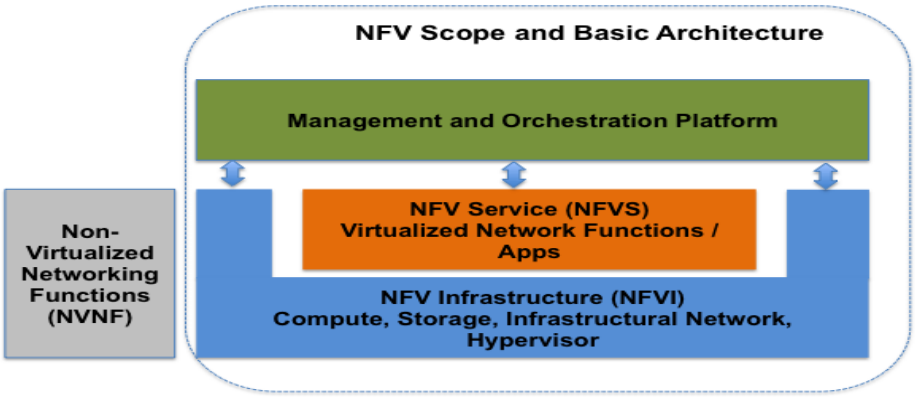


Figure 74: Network Functions Virtualization (NFV) Basic Architecture

A mapping and interworking of UMF functions and mechanisms to Network Functions Virtualization Architecture is presented in the following figure.

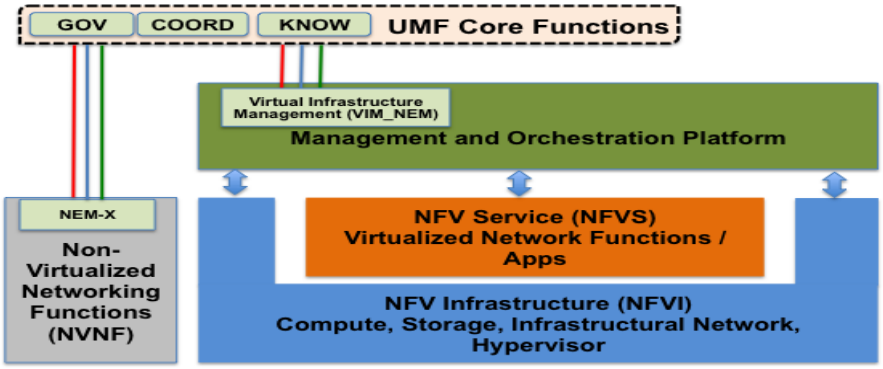


Figure 75: UMF Mapping to NFV

UMF would add autonomicity characteristics to the basic NFV architecture and it would also provide a management integration framework for the management of NV with Non NV Networking Functions.

6. Conclusion

Unified Management Framework (UMF) is 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 interworking of autonomic functions (e.g. Network Empowerment Mechanisms - NEMs).

UMF aims also the migration from an ecosystem of separate autonomic functions towards a coordinated arrangement of AFs as represented in the following figure:

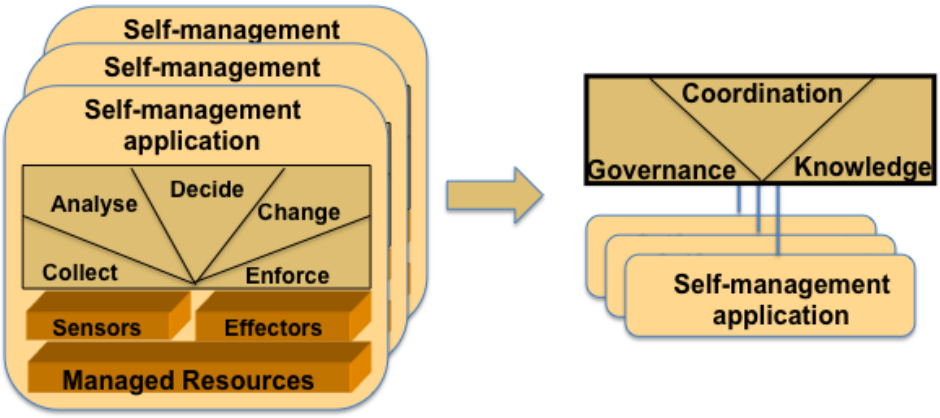


Figure 76 Migration from separate control loops to a coordinated arrangement of multiple control loops

Deliverable D2.4 "UMF Specifications – Release 3" provides a consolidated and wide-ranging functional specification of the UMF, including the detailed design specification of the UMF core functionality, the relevant interfaces and the mechanisms to support the main functions of the core blocks, the life-cycle management of the autonomic management applications – the Network Empowering Mechanisms.

UMF as a management framework is based on three main functional blocks namely, Governance, Coordination, Knowledge and the interworking with the autonomic management applications – the Network Empowering Mechanisms.

The UMF Knowledge block (KNOW) plays the role of information / knowledge indexing collection, aggregation, storage/registry, knowledge production and distribution across all UMF functional components. 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. Its main functionality includes: Orchestration, Conflict identification, Optimization and Conflict Avoidance. The role of the Governance block in response to the need of the human network operators is to supervise and control of the behaviour of the underlying autonomic functionalities (NEMs) and the UMF core blocks.

The three blocks together are managing the autonomic functions named Network Empowerment Mechanism (NEM). In this context, a set of interworking definitions for the NEMs are introduced and used for the full description of their lifecycle. 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 described.

The interworking of UMF with emerging network infrastructures, SdN (Software Driven Networks) and NFV (Network Functions Virtualisation), demonstrates that the incorporation of UMF and NEMs to those can enhance them with self-management capabilities.

The applicability of UMF with respect to 3GPP, eTOM, ETSI AFI GANA, IRTF Autonomic SDN, and ITU-T Future Networks Group architectures would drive the standardization work and further refinements of the UniverSelf solutions.

References

[1] UniverSelfProject Deliverable D2.2 – UMF specifications release 2.

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

[3] Java Specification Request 8 (JSR 8) Open Services Gateway Specification

[4] UniverSelf Project milestones MS26 & MS27 Unification of the mechanisms embedding the UC method.

[5] 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

[6] S.Davy, B.Jennings, J.Strassner, 'The Policy Continuum – A Formal Model', 2nd IEEE International Workshop on Modelling Autonomic Communications Environments, MACE 2007

[7] Enhanced Telecom Operations Map (eTOM) – The business process framework, ITU-T Recommendation M.3050.1.

[8] A. Galani, N.Koutsouris, K. Tsagkaris, P. Demestichas, B. Fuentes, C. Garcia Vazquez, G. Nguengang, "A Policy based Framework for governing Future Networks", 4th IEEE International Workshop on Management of Emerging Networks and Services (MENS 2012) at the IEEE Global Communications Conference (GLOBECOM 2012), December 2012, Anaheim, California, USA

[9] Steven Davy thesis, Harnessing Information Models and Ontologies for Policy Conflict Analysis (2008)

[10] UniverSelf Project, Deliverable D4.3, "Assessment Results of Trust in Autonomics Release 1", October 2012, available at http://www.univerself-project.eu/technical-reports [last accessed: February 2013]

[11] 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.

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

[13] TeleManagement Forum, SLA Management Handbook

[14] Galis, A., Denazis, S., Brou, C., Klein, C. "Programmable Networks for IP Service Deployment" ISBN 1-58053-745-6, pp450, June 2004, Artech House Books, www.artechhouse.com/Default.asp?Frame=Book.asp&Book=1-58053-745-6

[15] A.Galis, J. Rubio-Loyola, S. Clayman, L. Mamatas, S. Kukliński, J.Serrat, T. Zahariadis « Software Enabled Future Internet – Challenges in Orchestrating the Future Internet » - MONAMI 2013- 5th International Conference on Mobile Networks and Management, Cork, 23-25 September 2013 ; http://mon-ami.org/2013/show/home and A.Galis presentation at the 3rd ETSI Future Network Workshop 8-11 April, Sofia Antipolis ; http://www.etsi.org/news-events/news/617-2013-fnt-intro

[16] Network Functions Virtualisation (NFV) – ETSI Industry Group http://portal.etsi.org/portal/server.pt/community/NFV/367 & white paper http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[17] GB921-eTOM_Core_Standards_Release_13-0

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

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.*

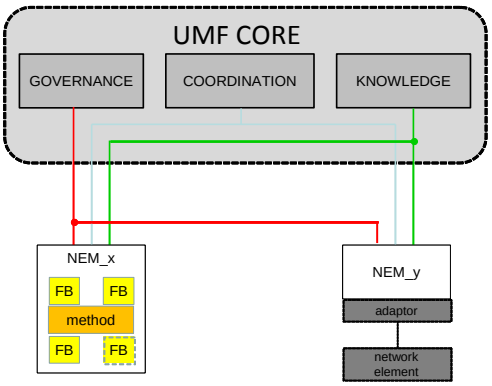
NEM manifest – *it describes a given NEM class. This description provides guidance to the network operator in order to install and configure an instance of this NEM class – the goal of a NEM manifest is similar to a datasheet). This description is issued by the NEM designer towards network operators.*

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 – A *software component developed by UniverSelf in order to provides to the NEM developer the UMF interfaces and the skeleton for 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, as the sepcifications of these 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.*