FP7 Information & Communication Technologies (ICT)

CONSERN

Deliverable D4.2

Distributed Self-Growing Architecture and Interface Description.

Document Number:	INFSO-ICT-257542/CONSERN /WP4/D4.2/300411
Contractual Date of Delivery:	30/04/2011
Authors:	Gunnar Hedby (Editor), George Koudouridis, Athanasios Makris, Alexandros Kaloxylos, Apostolis Kousaridas, Bernd Bochow, Marc Emmelmann, Markus Mueck, Makis Stamatelatos
Workpackage:	WP4
Distribution / Type:	PU (Public)
Version:	1.0
Total Number of Pages:	75
File:	CONSERN_D4 2_v3.0.docx

Abstract

This Deliverable, D4.2, on "Distributed Self-Growing Architecture and Interface Description" provides a proposal for a CONSERN self-growing architecture and describes its components and interfaces on a high-level. The architecture builds on the requirements that were derived in the earlier deliverables D1.1 and D4.1 and will be used and refined in future CONSERN research.

Executive Summary

The main goals in [1] set up for the CONSERN task T4.2 are:

- Elaborate on the outcome of T4.1 to set a foundation for subsequent implementations,
- Based on the functional requirements derived in T4.1, provide formal specifications for the functional entities and their relationship and other elements in order to specify a concise framework,
- Specify and design a dynamic cognitive control architecture suitable to “survive” major changes in underlying network topologies and capable to extend and learn from these changes,
- Propose and provide formal specifications for interfaces between the cognitive functions allowing to dynamically add or remove functionality and to integrate these into an enclosing architecture.

With the proposed self-growing architecture this deliverable has provided initial information to all of these goals. Obviously the research made during the very first year of the CONSERN project cannot provide a detailed architecture that can be used for implementation but it is felt further refinements of the model will come closer to this goal.

Contributors

First Name	Last Name	Company	Email
George	Koudouridis	HWSE	george.koudouridis@huawei.com
Gunnar	Hedby	HWSE	gunnar.hedby@huawei.com
Athanasios	Makris	NKUA	tmakris@di.uoa.gr
Alexandros	Kaloxylos	NKUA	agk@di.uoa.gr
Apostolis	Kousaridas	NKUA	akousar@di.uoa.gr
Makis	Stamatelatos	NKUA	makiss@di.uoa.gr
Bernd	Bochow	Fraunhofer	bernd.bochow@fokus.fraunhofer.de
Marc	Emmelmann	Fraunhofer	marc.emmelmann@fokus.fraunhofer.de
Markus	Mueck	IMC	Markus.Dominik.Mueck@intel.com

Acronyms

Acronym	Meaning
API	Application Programming Interface
AUC	Autonomic Control function
BSS	Basic Service Set
CCE	CONSERN Cognitive Engine
CE	CONSERN Entity
CCE-CCE	An interface between two CONSERN Cognitive Engines
CCE-CF	An interface between a CONSERN Cognitive Engine and a Functional Unit
CE-CE	An interface between two CONSERN Entities and is a superset of the CCE-CCE interface.
CE-F	An interface between a CONSERN Entity and a Functional Unit and is a superset of the CCE-CF interface.
COM	Communication Services
COP	Cooperation function
CSC	CONSERN Sensor Coordinator
CPM	CONSERN Policy Manager
CCG	CONSERN Configurable Gateway
FU	Functional Unit
IDL	Interface Definition Language
KBO	Knowledge Base Ontology
RDF	Resource Description Framework
SGN	Self-growing function
STA	Station
SysML	Systems Modelling Language
TRA	Translation Unit
UML	Unified Modelling Language
WSN	Wireless Sensor Network
XML	Extensible Markup Language

Table of Contents

List of Figures

List of Tables

1. Introduction and Outline

The CONSERN project aims at developing and validating a novel paradigm for dedicated, purpose-driven small scale wireless networks and systems characterized by a service-centric evolutionary approach introduced here as an energy-aware self-growing network.

Self-growing capabilities are enablers for a novel type of network flexibility. They allow optimising a heterogeneous collection of network nodes or sub-networks to be dedicated to a specific optimisation target ("purpose") temporarily and on-demand. From a technical perspective this approach relies on node and network reconfigurability for achieving adaptability to multiple applications. From a complementing business perspective this approach facilitates addressing multiple niche markets / vertical markets utilizing a single line of hardware/software developments.

In this deliverable a self-growing architecture model is defined based on the requirements and refinements thereof identified in [2] and [3]. The focus is on providing initial informal specifications for interfaces, primitives, functional requirements, functional entities and their relationship in order to specify a concise framework. The architecture presented shall be seen as being logical where the entities, when implemented in hardware, may be distributed into different nodes. The architecture model comprises two classes of CONSERN modules, the CONSERN Cognitive Engine (CCE) module and the Functional Unit (FU) module.

The FU is the module interacting with the environment by providing the needed functionality e.g., context provisioning and aggregation, communication service provisioning, knowledge provisioning, decision making & learning, and control & configuration services.

The CCE is the module initiating the execution of the functions to be performed by the FU and may comprise up to three levels of functionality, the administrative, operational and functional level. Depending on the role of a CCE it may include all these levels or only some of them. The functions provided by a CE map where applicable to those of the FU but also contain parts which are only applicable for communicating with other CEs. For information sharing each level of a CCE is able to communicate with its corresponding peer in another CCE via services provided by a dedicated communication module (COM) within the CE.

This document has the following outline:

In chapter 2, the self-growing requirements from the earlier delivery D4.1 [3] are revisited and a mapping to CONSERN functional entities is made. This mapping is later on used to propose a self-growing architecture which is the main goal of this deliverable.

In chapter 3, descriptions of two classes of basic modules (the CCE and the FU) are introduced. It is further discussed how the functions derived in chapter 2 can be mapped onto these modules. Due to the idea that self-growing networks may change their purpose (intentionally or unintentionally) during their life time the flexibility of the functions description format is emphasised. In addition, a suitability analysis, for the use cases derived in D1.1 [2], to demonstrate the full self-growing is given. The chapter concludes by providing a methodology for specifying the logical interfaces between CCEs and FUs.

In chapter 4), a self-growing stratum architecture model constituting the CONSERN project is given. The layers of cognitive control are: the Functional, the Operational and the Administrative layer and depending on its realisation a CCE may contain all or parts of these layers. The chapter also defines high-level interfaces between these layers and between networks comprising a more or less complete self-growing architecture. Finally, the chapter summarises the applicability of the layered self-growing architecture objectives to the use cases derived in [2].

In chapter 5), the CONSERN functional architecture is presented. The architecture is the realisation of the architectural model presented in chapter 4) and is designed to fulfil the requirements presented in chapter 2. A CE at least comprises three functional components, the CONSERN Cognitive Engine, the Communication Services and the Translation function. Also a formal overview of the Functional Unit (FU) functions is provided in the form of a UML model.

In chapter **Error! Reference source not found.**, the functional components which build the CONSERN system are described, the CONSERN Entity (CE), the CONSERN Policy Manager (CPM) and the CONSERN Sensor Coordinator (CSC). The CE is a logical entity that based on policy rules and previous actions decides on its next action in an iterative way. The action can be to control, or to communicate and negotiate information to, other CEs. The CPM provides the operator's policies that control a CE. The CSC controls a set of sensors and coordinates the information received from them. This is needed since there may be a huge number of sensors in a network. Finally, the CCG provides the interconnection of a CONSERN-enabled network with legacy networks/systems. Notice, that the CPM, CSC, and CCG are different realisations of the generic CE in the sense that their internal structure can be the same as the CE, but they have different scopes of operation.

In chapter 7, the interfaces needed for cooperation between CEs (CE-CE) and between a CE and a FU (CE-F) are defined. The CEs involved in a network may be of different types and are not always implementing a full self-growing functionality. The CPM, the CCG and the CSC are examples of two such types. Therefore an important part of the interfaces is the subset of the general interface CE-CE used to cooperate between such partially implemented CEs. Finally the chapter presents an idea on the data structure for some self-growing descriptors.

Chapter 8 details basic service types as they are introduced in chapter 7 and provides Message Sequence Charts (MSC) showing the foreseen interactions between CONSERN Cognitive Engines.

2. Review of CONSERN Framework

2.1 Architectural Requirements for Self-Growing

In D1.1 [2] a set of system requirements, as identified in conformity with the definition of the self-growing concepts, have been further classified as functional and non-functional requirements. Functional requirements capture the intended behaviour of the system which can be expressed as functions the system is required to perform, whereas non-functional requirements specify overall system characteristics and constraints which the system needs to satisfy.

The following list presents the initial set of functional and non-functional requirements:

- **Functional Requirements**
  - Monitoring and measurements. Measurements are foreseen to be made of:
    - Energy consumption in participating networks,
    - End-to-end delay across network boundaries,
    - Properties of communication path (delay, PER, etc.),
    - Network density,
    - Change of the desired network purpose.
  - Information and knowledge dissemination and sharing. This is foreseen between:
    - CONSERN Entities (CE),
    - CONSERN Entities and FUs.
  - Learning
  - Network node detection and integration. This includes:
    - Node and network discovery in the area (mainly for WSN),
    - Evaluation regarding the type of measurements provided by WSN networks,
    - Dynamic integration of additional network nodes, networks, and/or WSNs.
  - Decision Making:
    - Cooperative/ autonomic decision making on optimal configuration,
    - Cooperative relay power allocation,
    - Content-oriented traffic offloading.
  - Optimization action application:
    - Adjust number of hops,
    - Adjust number of cross network transitions,
    - Adjust node spread and location in the service area,
    - Adjust number of network purposes provided
    - Normalize load distribution,
    - Reconfigure network elements,
    - Reconfiguration capability at PHY and MAC layer,
    - Device remote firmware update.
- **Non-functional Requirements** impose specific constraints in the system behaviour enabled by the functional requirements. In this sense, the designed functionality needs to ensure operation under the following constraints:
  - Low Power,
  - Seamless,
  - Dynamically Adjust,
  - Dependability and Efficiency.

In this approach the functional requirements drive the functional architecture definition that will be evaluated in the scope of the non-functional requirements/constraints (as they are detailed in [2]) within simulation and prototyping frameworks as planned.

2.2 Mapping of functional requirements to functions and functional entities

Based on the analysis in [2] the following set of functions (in different levels of detail) was derived in [3]. From these functions this deliverable proposes an architecture for self-growing, see chapters 4 and 5.

- Sensor coordination: A function responsible to collect information from a set of sensors and make it available to other functions,
- Network configuration: A function responsible for the configuration and operation of network elements,
- Information exchange: A function responsible for information exchange between the network components, such as Sensors and/or a Sensor Coordinator, relay nodes,
- Knowledge sharing: Decision and operating command distribution among network elements on balancing autonomic capabilities, cooperative operation, etc.,
- Information awareness/learning: Collecting and learning the communication environment information such as network topology, channel information, traffic load, neighbour nodes statistics and etc.,
- Decision making: Distributed operation decision (cooperative decision) making based on learned information.

The design of the proposed architecture is driven by the functional requirements extracted in [2] and [3]. Requirements that were not previously identified in those deliverables but came up during the design process are identified and included as well. Based on the above list of general functions a set of functional entities can be identified as an initial input to the CONSERN architecture. In this context a functional entity can be broadly defined as a unit of software and/or hardware that implements a specific function (for the system). The mapping between the extracted functional requirements and the proposed functional entities is presented in Table 2-1. Chapter 6 provides more detailed information of the CONSERN functional entities.

Functional Requirements	Functional Entities / CONSERN Entity functions
Monitoring and measurements	Sensor Data Aggregator CONSERN Sensor Coordinator (CSC) Monitoring function
Information and knowledge dissemination and sharing	Policy Provider CONSERN Policy Manager (CPM) Knowledge Base Ontology (KBO) CONSERN Sensor Coordinator (CSC) Sensor Data Aggregator
Learning	Learning function
Decision Making	Self-growing function Cooperation function Autonomic Control function

Network node detection and integration	Self-growing function Cooperation function Knowledge Base Ontology (KBO)
Optimization action application	Execution function Translation function
Network configuration	Self-growing function Cooperation function Autonomic Control function
Interconnection with legacy networks	CONSERN Configurable Gateway (CCG)

Table 2-1: Mapping of functionalities to functional entities.

In the remainder of this section a description of the main functions, the corresponding functional entities and the functional requirements which are fulfilled are presented for the sake of completion. It has to be noted that these descriptions are input to the architecture specifications and subject to modifications along the design process. Although these function descriptions will be defined - and in some cases redefined -in later chapters, the initial ideas are summarised in the following paragraphs.

Monitoring is performed in different elements of the network because of the variation in the parameters that are being monitored. Also, aggregation of the monitored data is necessary in order to avoid high control overhead in the network. Based on these requirements, the functional entities that are designed to provide the monitoring functionality are the Sensor Data Aggregator (first level of aggregation to sensor reported data), the CONSERN Sensor Coordinator (second level of aggregation, pre-processing), and the Monitoring function (third level of aggregation, post-processing, correlation).

Information and knowledge dissemination and sharing is provided by the CONSERN Policy Manager in the sense that intelligence and high-level policies are injected in the network by the Policy Provider through the CONSERN Policy Manager. Also, the Knowledge Base Ontology is used to store all types of information (static, dynamic, knowledge).

The Learning function generates knowledge in the system based on a set of available information and performed management actions.

Network node detection and integration is a part of the Self-growing and Cooperation functions. The self-growing paradigm is realized by the CONSERN Cognitive Engine which provides all the necessary mechanisms for enabling self-growing and self-x management operations.

Decision-making is the procedure that generates re-configuration actions based on the current state of the network and a set of rules that define its optimum state. This functionality is provided by the Decision function.

Optimization action application refers to the translation and application of abstract re-configuration commands to vendor/hardware specific re-configuration actions. This functionality is provided by the Execution function which takes the decisions from the Decision function and generates abstract re-configuration commands, and the Translation function that translates these commands to hardware/vendor specific actions.

Network configuration includes multiple levels of configuration of the network components to achieve self-growing, self-configuration, self-healing, and self-optimization. These functions can be performed for different types of networks and targeting different goals. The CONSERN Cognitive Engine and its sub-entities provide all these different levels of the self-growing and self-x network management.

3. Realisation of Self-Growing Enablers

3.1 Modules and Functions enabling Self-Growing

Enabling self-growing involves in general two classes of modules: a **consern cognitive engine** (CCE) and a **functional unit** (FU). The CCE realizes the cognitive decision capability for self-growing networking; it thereby gives guidance to the FU on how to behave, and receives context information from the FU. In the relation between the two modular classes, “guidance” comprises instructions from the CCE to the FU for coarse, rather unspecific range of operation (e.g. setting ranges of permissible frequency bands to operate in, or allowing a local decision at the FU for setting its transmission powers as long as it is below a certain threshold), up to providing stringent, mandatory rules to be obliged by the FU e.g., set transmission power to a given, fixed level, or enabling/disabling certain interfaces for communication with a given other node. In its utmost controlling form, such a rule would immediately and unconditionally set a fixed configuration of the FU.

3.1.1 *Functional Unit Module*

The **functional unit** is the only module interacting (partially on behalf of the cognitive engine) with the environment. It can operate as an actor within the system e.g., providing functionality to route IP traffic between two networks by communicating with another node, or as a sensor providing temporal- and special-specific context information.

As a module enabling self-growing, the FU provides the following functionality: a) context provisioning and aggregation, b) communication service provisioning, c) knowledge provisioning, d) decision making & learning, e) control & configuration services. An initial set of functions per functionality group is given below:

1) Context provider / aggregator
    a) Scanning, sensing, node detection: Detection of other technologies or even specific nodes within the proximity of the scanning node,
    b) Measurement function: capability to report on a given property of the node e.g., experienced noise level, remaining battery life time, experienced packet loss, etc,
    c) Data aggregation / fusion: capability to process or combine sets of measurements according to a given objective e.g., single controller receiving and aggregating sensing information from several attached sensors,
    d) Service detection: detection of services and capabilities of other nodes within the (communication) vicinity of the node. This includes the capability to discover nodes instantiating a CE to connect to,
    e) Service and capability description: capability to inform other entities of the services and capabilities the node may provide e.g., self-description of node’s (hardware) capabilities, current configuration and limits via the Resource Description Framework (RDF).
2) Communication service provider
    a) Service announcement: Provisioning of services, the node may provide for a given configuration,
    b) Addressing & routing: Functionality provided by the FU to connect two networks possibly fulfilling different purposes, and to address nodes within the other network,

   c) Message transport service: ability to send/receive a message from another node considering the following options: unicast/broadcast, reliable/unreliable, message-/flow-based communication,
   d) Security services: Association, authentication, etc. of the FU with communication partners.
3) Knowledge provider (only applicable, if FUs locally employ rules of operation e.g., rule sets assuring operation according to a previous certification process)
   a) Rulebase,
   b) Policy provider.
4) Decision making & learning (only applicable, if FUs locally employ decision engines for realizing certain functions)
   a) Algorithmic learning,
   b) Heuristic / fuzzy learning,
   c) Rule-based (structured) decision making,
   d) Ontology-based (logical) decision making,
   e) Policy enforcement,
   f) Context filtering.
5) Control & Configuration service
   a) Node configuration: capability to change the configuration of the node to a given value or to be within given rules of operation,
   b) Network topology configuration: capability to change the mode of operation from e.g., acting as a node within a hierarchical topology (i.e. 802.11 STA connecting to an infrastructure Basic Service Set) to an ad-hoc, multi-hop operation (i.e. 802.11 STA acting as a relay in a meshed network),
   c) Context provider / sensor configuration: capability to direct the configuration of sensing units controlled by the FU,
   d) Communication system service configuration: configuration of communication service capabilities,
   e) Knowledge provider management services: Communication with the knowledge provider management service,
   f) Cognitive control and management: communication with the cognitive control and management functionality of the FU.

The actual parameters and responses, or even the availability of certain functions depend on the capabilities of the device instantiating functions of the functional unit. Hence, **one function is mandatory to implement: reporting the list of capabilities** of the functional unit (service and capability description). Such report shall at least include the physical capabilities of the functional unit (e.g. available technologies for communication, configuration options and their parameter range, support of dynamically adding functionality to the functional unit, number of parallel available transceiver chains for different technologies, etc.) and available functions. The description of the capabilities shall be in a structured and standardized format. Due to the evolution of networks as seen by the self-growing approach, such description format shall specifically support the evolution of schemas over time without requiring all the data consumers to be changed and should inherently facilitate data merging even if the underlying schemas differ. Employing the Resource Description

Framework (RDF) [7] easily ensures those requirements. In addition, devices instantiating the functional unit are mandated to implement one additional function: the service detection function allowing the detection of available cognitive engines within the FU's vicinity. Obviously, in order to establish a direct information exchange between FU and CCE, message exchange between the two modules has to be enabled by the network or respectively other dedicated units within the CE.

3.1.2 *Cognitive Engine Module*

The **consern cognitive engine** initiates the execution of those functions. Hence, a communication between CE and FU is required since the CCE may not directly interact with the FU except for using CE-F interfaces. A technology independent addressing scheme shall be used to address entities instantiating the cognitive engine and the functional unit. Given the wide acceptance of the Internet Protocol and without loss of generality, we assume an IP-based communication protocol between the two modules herein. Using the IP-based protocol suites gives the freedom during the protocol design phase to build upon a variety of standardized transport protocols for streaming / packet oriented delivery with or without mechanisms for coping with packet loss and ordering.

Cognitive engines enabling self-growing may enable synergies among different network providers, within one operator's domain among nodes, and directly between nodes without the operator's control by having cognitive capabilities on individual devices. Correspondingly, we find CCEs at the administrative, at the operational, and at the node (functional) level. Considering those three levels gives a degree of freedom in designing an architecture supporting self-growing; CCEs may exist in one compact module (coping with the administrative, operational, and functional levels) or may be distributed within the system having a separate CCE per level of self-growing. Accordingly, functions of each sub-module have to be accessible by other sub-modules and each sub-module has to be capable of either directly or indirectly accessing functions within functional units. As a higher layer CE's sub-module may indirectly access a FU via lower layer CE's sub-modules, functionality provided by CE's sub-modules has to encompass functionality of FUs.[1]

Accordingly, the functionality of the CE module(s) resembles those of the FU though the semantics differ in certain cases:

1) Context provider / aggregator
    a) Scanning, sensing, node detection: detection of other available nodes or networks,
    b) Measurement function: capability to report on given properties of the cognitive engine e.g., information on the rule set the cognitive engine currently operates on.
    c) Data aggregation / fusion: capability to receive information from other cognitive engines (aggregation process) in order to obtain a condensed set of information (fusion process) e.g., via filtering processes,
    d) Service detection: detection of services offered by other FUs within the vicinity of the CE or of services (purposes enabled by the network under the control of other CEs) that may be realized,
    e) Service and capability description: capability to inform other CEs of the services and capabilities the CE may provide e.g., self-description of CE's capabilities to enable certain purposes of a network, current configuration, etc. via RDF.

[1] Note that this requirement towards the functionality of each module needs to be later on reflected in designing appropriate interfaces. The implementation of such interfaces might also distinguish between the level of interaction among modules and different semantics of the function applied at a given level. Also, this function and modulation concepts allows for implementations in which FUs internally employ (hidden) CCEs for the realization of certain functions, as well as CEs internally employing (hidden) FUs.

2) Communication service provider
    a) Service announcement: announcement of the presence of the CE itself as well as of services/purposes of the network that the CE may enable,
    b) Addressing & routing: capability to communicate with other modules locally and across network boundaries,
    c) Message transport service: ability to send/receive a message from another node (realizing CEs or FUs) considering the following options: unicast/broadcast, reliable/unreliable, message-/flow-based communication,
    d) Security services: establishing of a level of trust between communication entities (CE-CE or CE-F).
3) Knowledge provider
    a) Rulebase: provisioning of rule base the CCE operates on,
    b) Policy provider: provisioning of policies the CE operates on and of the source of the policy set. This may include specific policies for CCE operation as well as policies guiding the operation of other modules within the CE.
4) Decision making & learning: capabilities associated with the internal functionality of the CCE
    a) Algorithmic learning,
    b) Heuristic / fuzzy learning,
    c) Rule-based (structured) decision making,
    d) Ontology-based (logical) decision making,
    e) Policy enforcement,
    f) Context filtering.
5) Control & Configuration service
    a) Node configuration: configuration of individual FUs or other connected CEs,
    b) Network topology configuration: configuration of network topology by instructing several FUs to act in a given manner,
    c) Context provider / sensor configuration: ability to provide context information on the current rules of operation of the CE as well as instructing other modules (CEs or FUs) to provide given context information,
    d) Communication system service configuration: configuration of communication service capabilities,
    e) Knowledge provider management services,
    f) Cognitive control and management: management and configuration of cognitive control capabilities of the own CE as well as how to manage other modules in the system.

3.2 Suitability of Use-Cases for Demonstrating Full Self-Growing

A “full self-growing system” supports self-growing capabilities at the administrative level, at the operational level, and at the functional (node) level. Table 3-1 summarizes the CONSERN Use Cases as defined in [2] and at which level they include self-growing capabilities.

Use Case	Characteristic of Cognitive Engine: provisioning of self-growing at the		
	Administrative Level	Operational Level	Functional / Node Level
UC-01: Energy Optimization in a moving vehicle with capacity and coverage limits		yes	
UC-02: Energy Optimization in an Office environment under coverage constraints	yes	yes	yes
UC-03: Energy Optimization for Self-Growing Office environment under coverage and capacity constraints		yes	yes
UC-04(1): Network reconfiguration following the introduction of new nodes	yes	yes	
UC-04(1): Network reconfiguration following the introduction of new nodes	yes		yes
UC-05: Switch on-off of nodes for Energy Efficiency in Heterogeneous Networks		yes	
UC-06: Cooperative DAS nodes configuration	yes	yes	
UC-07(1): Cooperative relay for Energy Efficiency	yes	yes	
UC-07(1): Cooperative relay for Energy Efficiency	yes		yes
UC-08: Energy-aware end-to-end delay optimization	yes	yes	yes
UC-09: Purpose-driven network reconfiguration during an emergency situation	yes	yes	yes
UC-10: Cognitive Coexistence and self growing for white space operation		yes	yes
UC-11(1): Energy optimization of co-located wireless networks in a home/office environment	yes	yes	
UC-11(1): Energy optimization of co-located wireless networks in a home/office environment	yes		yes
UC-12: Self-adaptation of a reconfigurable wireless terminal			yes
UC-13: Home Monitoring Energy Optimization		yes	yes
UC-14: Cooperation Enablers in Home Gateway Environments		yes	yes
UC-15: Dynamic Meeting Setup Flexible Office/Building Environments	yes	yes	yes
UC-16: Collaboration of Different Technologies		yes	yes
(1) The cognitive engine does not need to provide operational and functional characteristics at the same time for this use case.			

Table 3-1: Use-Cases providing Self-Growing at administrative, operational, or functional/ node level.

The analysis shows that four CONSERN use cases are best-suited to simultaneously demonstrate the full self-growing capabilities:

- UC-02-NKUA: Energy optimization in an office environment under coverage constraints,
- UC-08-Fraunhofer: Energy-aware end-to-end delay optimization,
- UC-09-Fraunhofer: Purpose-driven network reconfiguration during an emergency situation,
- UC-15-Fraunhofer: Dynamic Meeting Setup Flexible Office/Building Environment.

Another set of use cases allows as well demonstrating all functional levels of self-growing, given that all “flavours” of the use cases are considered. Those use cases are:

- UC-04-IFX: Network reconfiguration following the introduction of new nodes,
- UC-07-HWSE: Cooperative relay for energy efficiency,
- UC-11-IBBT: Energy optimization of co-located wireless networks in a home/office environment.

Given that the decision within those use cases to implement self-growing capabilities either at the functional or operational level (in addition to the administrative level), demonstration of self-growing at the function/administrative and at the operational/administrative level should be feasible inducing slight changes in the implementation of either use case. Accordingly, WP-4 suggests to WP-5 to consider a subset of the above listed seven use cases for demonstration purposes. The criteria for choosing a subset are to be argued within WP-5 and may consider ease of implementation or short-term commercial relevance.

3.3 Architectural constraints for Self-Growing

The dynamic cognitive control architecture developed in CONSERN shall be capable to “survive” major changes and/or extensions in underlying network topologies; this requires the architecture to avoid single points of failure in the design. Resulting constraints for the architecture include provisioning of redundancy or **distributing cognitive control functionality across several CONSERN entities** (on different network nodes). Primarily, CONSERN follows the latter approach as functionality of the cognitive engine can be grouped into cognition and control on a functional (node), operational (network), and administrative (management by network operator) level. Hence, functions of a full self-growing node are under the control of a (local) cognitive engine. Upon a node entering a network i.e., staring to participate in communication, its functional cognitive engine makes itself (and thereby implicitly the availability of the node under control of the cognitive engine) visible to the cognitive engine at operational (network) level. This requires the self-growing CONSERN architecture to support **service discovery or service announcement** (of self-growing capabilities) to be provided by a functional entity specialized in communication services between CONSERN entities (CEs) or service entities within CEs. Hence, the same provided service detection scheme shall allow cognitive engines at an operational level to find and connect to cognitive engines at administrative level.

A sudden change in the underlying network topology may cause an interruption of the former communication path between cognitive engines at each level; also new nodes may join the network. Loss of communication as one indication of topology change is pair-wise detected by participating cognitive engines using the service discovery or service announcement features to re-establish hierarchical communication between cognitive engines on the functional, operational, and administrative level.

If a change in topology causes networks to merge, several cognitive engines at the operational and / or administrative levels might be discoverable and announce their "willingness" to take cognitive control of the system. The CONSERN architecture shall support the **detection of redundant cognitive functionality**. This allows redundant engines to negotiate which one takes temporal control of the system and which engines should turn into a "dormant" state. Such detection can be enabled by a dedicated functional entity specialized in providing communication between CONSERN entities and in providing service discovery or service announcement.

In case of network partitioning, cognitive engines on the operational and/or administrative levels might not initially exist in network fragments. The CONSERN architecture shall hence support **detecting the absence of** such **required cognitive functionality** to allow appropriate actions to be taken: for example, cognitive engines discovering missing cognitive functionality may cause an **instantiation of a new cognitive engine on selected network nodes** based on existing CE prototypes. Alternatively to instantiation, dormant cognitive engines can reactivate themselves upon detecting the absence of functionality that they can provide.

Any merge or partitioning of nodes or (sub-)networks results in a change of interacting cognitive control engines and in case of functional control engines in a change of available functionality in the self-growing network. Such loss or gain in (network) functionality is detectable by involved control entities. Associating a network / topology change with the resulting change in (network) functionality is one approach to learn how changes affect network capabilities.

Apart from service / capability discovery to be provided by underlying network services in the CONSERN architecture, interfaces between cognitive engines and between cognitive engines and functional units on network nodes should

- Compass a core set of functions via statically defined service primitives,
- Allow for node / technology specific implementation of means to achieve actions requested by service primitives, and
- Enable dynamically enabling or removing (core) functionality on nodes.

This can be achieved by separating the initiation of a function call as defined by the interface between units in the CONSERN architecture from the implementation of its functionality. Implementation of functionality can be stored in form of an implementation repository which can be extended by implementations of functionality which was previously not present in a particular CONSERN Entity. Means for dynamically adding (or removing) parts of the repository can be provided e.g., in combination of an entity specialized in providing communication services between CONSERN Entities and an entity managing the CONSERN Entity itself.

An open distributed object computing architecture building its functionality on top of OSI transport layer may be suitable. This decouples the self-growing functionality of a network from the actual realization of underlying network services such as addressing and establishing transport streams between involved entities. Also, service discovery / capability announcement can be assumed to be available in realizations of distributed object computing.

Without limiting the general applicability of self-growing capabilities to other realizations of distributed object computing, we build upon an object request broker based approach e.g., CORBA.

The interface between cognitive engines or between a cognitive engine and functional units realizes as a client-server relationship. This immediately allows device manufacturers to deploy a technology specific implementation of self-growing functionalities on their nodes which can be invoked (remotely) by cognitive engines.

Dynamically adding or removing such implementations of functions provided by the CONSERN Entity enables a CONSERN Cognitive Engine to provide various services; though the rules sets and policies being responsible for intelligently using those functional entities need to be (re-)configurable as well, in order to dynamically add or remove functionality of the CONSERN Cognitive Engine (CCE) itself.

The resulting functionality (i.e. querying rule sets controlling other CCEs and negotiating rule sets among CCE itself being decomposable into setting and removing (agreed) rules) is inherently part of a CCE as part of the collaboration process among CCEs. Collaboration between CCEs enables deployment of rules, policy rules or general facts from one CCE to one or more neighbouring CCEs. CCEs situated in their geographical vicinity may receive remote knowledge at any time.

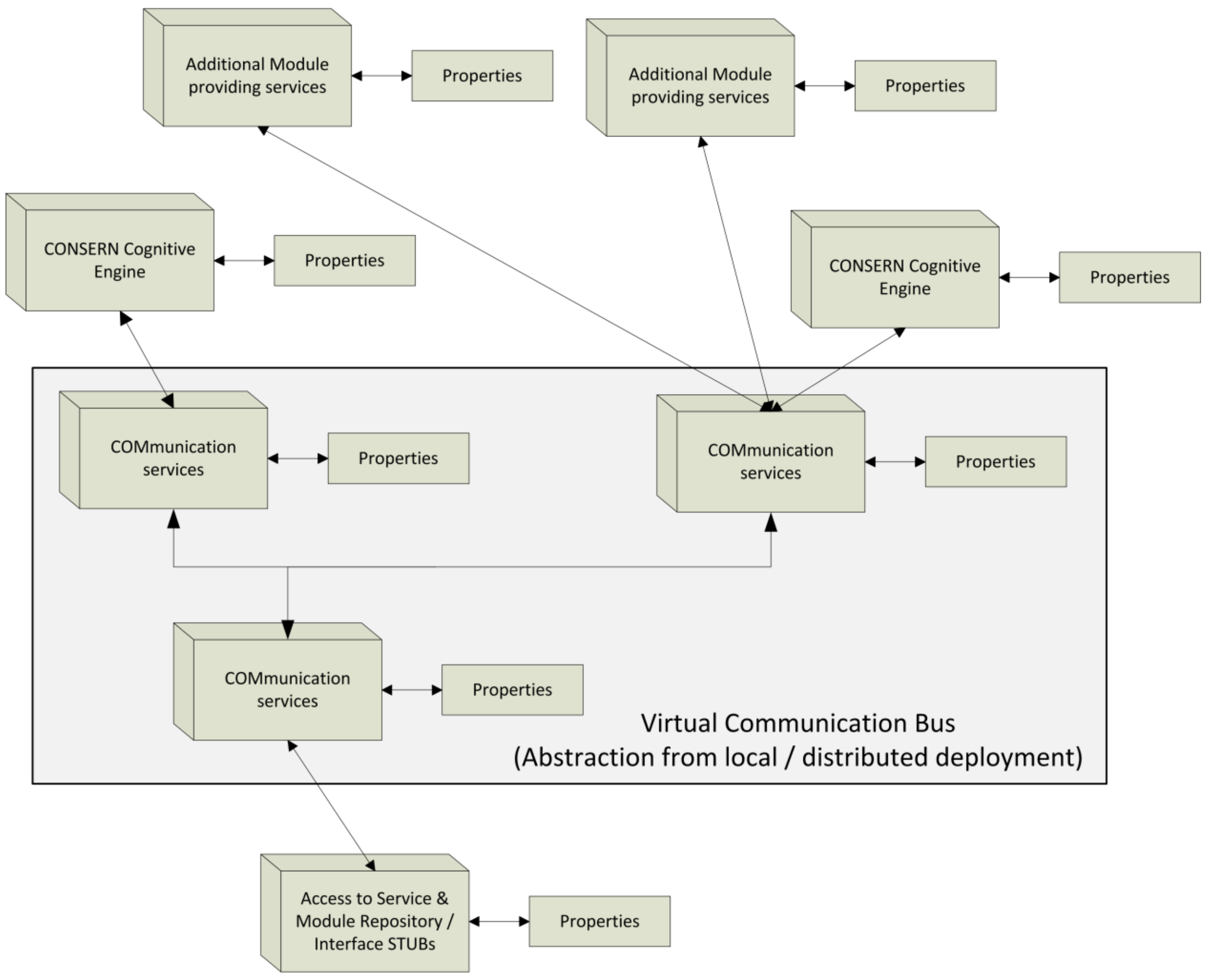
Additional Module providing services
Properties
Additional Module providing services
Properties
CONSERN Cognitive Engine
Properties
CONSERN Cognitive Engine
Properties
COMmunication services
Properties
COMmunication services
Properties
COMmunication services
Properties
Virtual Communication Bus (Abstraction from local / distributed deployment)
Access to Service & Module Repository / Interface STUBs
Properties

Figure 3-1: Key components for the CONSERN Self-Growing Architecture.

In order to realize the self-growing approach within CONSERN, the distributed object computing approach is implemented using CORBA. Interfaces between functional entities within a CONSERN Entity are specified using the Interface Definition Language (IDL). The services provided by a given functional entity (module) via its Service Access Point (SAP) interface are realized via the implementation of the service primitive as part of the CORBA client STUB paradigm. Calling entities using the interface simply invoke the service primitive name in form of a function call.

Providing empty hulls for the client-side implementation of the STUB has several advantages for advanced and rapid prototyping as well as for collaboration among project partners:

For rapid prototyping, a partner focusing on a specific functionality (e.g. the CONSERN Cognitive Engine) may implement himself a dummy version of the interface. This can be as simply realized as substituting actual functionality of a real world implementation by statically returning requested information or acting upon requested behaviour in a predefined manner. This approach is usable for providing predefined testing vectors to the implementation of a specific functional entity (e.g. CCE) for testing purposes.

For integration purposes, the (dummy) implementation of functionality within the repository may be replaced by the implementation of the client-side STUB coming from partners realizing the associated functionality.

Even if partners do not choose to implement (full) functionality within the CORBA framework, the implementation of the client-side STUB to be provided by each partner may be reduced to simply attaching their implementation to the integratable system by “forwarding” message calls to their implementation outside the CORBA framework. In this case, a simple, state-less message translation is provided.

In addition, the CONSERN Cognitive Engine will be developed to run within a virtual machine (VMware). Hence, cognition for self-growing may be directly added to any device running the VM and providing implementations of the client-side STUBs accessible via CORBA.[2]

3.4 Methodology for Specifying Interfaces between CCEs and FUs

3.4.1 *Description of Parameter for each Functionality*

Based on the initial set of functionalities listed in Section 3.1, a list of parameters per function has to be derived per functionality. Starting from this list, the following steps are proposed to generate each parameter:

- Decide on one or two use cases followed upon within CONSERN and use those to identify the parameters of each functionality required for this specific use case,[3]
- Describe the information elements corresponding to the identified parameters based on a formal parameter description, and
- Specify the encoding of information elements in a way that they can be used as function parameters in an API call as well as in the payload of a communication system message.

Since the interface specification must be open to future extensions of the parameter sets, extensibility of the descriptions provided as well as the overall set of parameters considered is mandatory.

We currently assume that all parameters can be expresses as:

- Fixed size basic (i.e. single value) parameters (numerical values or enumerations),
- Variable length octet (byte) strings or arrays (e.g. character strings or numerical arrays of basic parameters), or

[2] For a commercial adaptation of the project results, the code realizing the CONSERN Cognitive Engine may be ported to a target system such that it runs natively on the system avoiding the usage of virtual machine.

[3] Note: As the list of parameters per function is extensible, this approach does not constrain the interface specification but enables rapidly approaching a performance evaluation of the self-growing concept within the context of a given use case.

- Structured complex parameters consisting of an arbitrary collection of one or more of the latter two.

where encoding rules for each of those parameters may differ.

A **formal parameter description** must comprise a short informal summary of the meaning and purpose of the parameter and a unique name to be used as an unambiguous identifier throughout the next steps. The parameter description should provide a specification of the expected value range and the desired resolution. Unbounded parameters are possible but will result in a less efficient encoding.

Different value ranges might apply for the same parameter depending on the desired usage of the parameter in different contexts. Since a distinct unique name is required for each of these, this implies that a dedicated parameter description for each value range case or resolution case might be needed. A hierarchical naming would satisfy this demand with minimum overhead – the formalism used might denote inheritance from a general parent description wherever suitable.

Example 1:					
Name	Frequency	**Phys. Unit:**	Hz	**Extends:**	--
Desc.	Basic unbounded frequency parameter used as a template				
	Range (min/resolution/max):		0	--	--
Name	Frequency.scan	**Phys. unit**	Hz	**Extends:**	Frequency
Desc.	Frequency parameter used in spectrum scanning. Extends frequency parameter by an upper limit (max) and step size (resolution). Restricts start frequency (min).				
	Range (min/resolution/max)		1MHz	10kHz	10GHz

In case a parameter value range is non-contiguous or does not follow a simple creation rule, enumerations can be used to describe the parameter's valid value space. Enumerations based on simple numerical values are preferred for their straightforward implementation but complex enumerations based on structured complex parameters are also allowed. Each enumeration instance requires a unique name to be used as an unambiguous identifier.

Example 2:					
Name	Frequency.band	**Phys. Unit:**	GHz	**Extends:**	--
Desc.	Enumerated list of valid frequency bands each defined by upper and lower band limit.				
	Frequency.band.ch01		2.4095	2.4145	
	Frequency.band.ch02		2.4145	2.4195	

An **information element description** extends a parameter description by information required for transmission as a payload of a communication system message such as element size or identifier (format tag), or for formatting the parameter for API use e.g., element type assumptions. It introduces additional information describing the parameter to allow for efficient encoding and decoding the parameter for the intended use. It does not provide final encoding information such as binary representation (for communication system use) or element type (for API) use – this specification is left to dedicated encoding rules. For example, the information element description specifies a parameter as a variable size array of integer elements and determines the need for an integral size parameter, but does not specify how the array size or the integer representation of each element finally is encoded.

Example 3:					
Name	Frequency.band	**Id:**	37	**Size:**	2
Desc.	Defines a frequency band information element. Each information element consists of the encoded Id and two encoded numerical values giving upper and lower band limit. The parameter size given is the number of parameter values needed (thus excluding the parameter identifier).				
Name	Frequency.band.pre	**Id:**	38	**Size:**	1
Desc.	Defines a frequency band information element. Each information element consists of the encoded Id and a unique identifier for some predefined Frequency.band parameter already known to both transmitter and receiver. In this example enumeration values of Frequency.band (Frequency.band.ch01, Frequency.band.ch02 …) might be used for this purpose.				

The **encoding rules** for a parameter finally map the information element onto a binary representation. A huge number of options exist to describe this mapping e.g., UML, XML, ASN.1, CORBA IDL to name a few. Although encoding is not considered essential for the standardisation, it nevertheless should provide simple examples or should refer to an external document for defining the encoding of each parameter to assist developers. Alternatively, a simple formal description of the encoding rule might be given. The ASN.1 encoding control notation may serve as an example.

Example 4:	
Name	Frequency
Enc.	Double precision non-negative real number according to IEEE 754-2008 [5]
Note	API use only
Name	Frequency.scan
Enc.	32 bit 2's complement unsigned integer value (val) with Frequency [Hz] = val * 10.000. Min{val} = 1, Max{val} = 1000000, Resolution{val} = 1

3.4.2 *Encoding rules for API use and communication system use*

The specification of parameters should be suitable for both API calls and remote communication based on the same formal parameter description. In principle only the last step of parameter specification (i.e. setting encoding rules) depends on the intended use. A separate specification must be given for each intended use, for example, using ASN.1 for communication system use and as a "C" language struct for API use.

3.4.3 *Suggested message structuring*

If used in remote communications, function calls and their parameters must be enclosed in a transport envelope (message). Parameters then are carried by the message (message payload). The minimum information required for message transport (message header) is summarized by the following table. Additional management information might be needed.

Message format		
	Message header	
M	Protocol type	Identifies the message unambiguously
M	Protocol version	Version protocol used
M	Layer control information	Information required to ensure proper operation of end-to-end protocol such as, sequence number checksum, stream flags or similar.
M	Information source	Unambiguously identifies information originator. Maybe an address (e.g. next level address of transport system) or module id (CCE or FU). Both source and destination identification is required to enable bi-directional communication.
	Information destination	Unambiguously identifies the desired information destination (see information source for explanation).
O	Source address	Unambiguously identifies the message originator. Maybe different from the information source. If the transport layer cannot determine the source/destination address from the information source/destination, this information must be provided in order to establish a communication link between information source and destination. Maybe also needed if message forwarding is required. Both source and destination identification is required to enable bi-directional communication.
	Destination address	Unambiguously identifies the message destination (see source address for explanation).
	Message payload	
M	Payload type	Unambiguously identifies the message to the information destination. Currently this field is used to distinguish messages that carry control information from those carrying sensing information. This field may be extended to provide a priority level (please note that this is end-to-end/application relevant priority, transport system priority is layer control information). This field may be seen as a part of the address field since it can be used to reference a specific functional unit within a node instantiating several FUs or CEs. In that it is similar to a TCP/UDP port address.
M	Parameter Id	Unambiguously identifies the parameter. This field maps directly to the unique name given to the parameter in its description.
	Parameter value	Parameter value. May be a fixed size (basic) parameter value, a variable length octet (byte) string, an array of parameter values, or a complex structured parameter as given by the parameter description. Variable length parameter values must include a length/size specification.
O	Parameter Id	One or more optional parameters. May be appended by the information source as needed.
	Parameter value	

Table 3-2: Example of suggested message structuring.

3.4.4 *Suggested description of parameter structure for API use*

We suggest CORBA IDL for the description of parameters. The advantage of applying this particular description scheme lies within the ability of automatically generating an implementation framework in various languages. Apart from supporting capabilities to enhance nodes dynamically with additional functionalities during run-time via the CORBA brokerage approach, this reduces the need to manually implement communication between devices given they reside within IP networks.

4. Architecture Model

This section provides a description of the self-growing architecture first presenting a high level view to the layered cognitive control architecture in order to elaborate on the basic interfaces encountered i.e., the reference points of the architecture. Next, the architectural model is detailed further based on a perspective similar to the 3GPP UMTS stratum model [4] to emphasize on the distribution of functionality throughout the architectural model.

In Figure 4-1 the three layers of cognitive control are introduced as:

Functional: This layer considers network nodes or whole networks as a collection of functions. Two reference points constitute on this layer: a) the CCE-CF interface between the CCE and the FU, which can be understood as a control and configuration interface for reconfigurable device and b) the CCE-CCE interface between cognitive engines. The latter can instantiate as both an intra- or inter-layer interface.

Operational: This layer considers cognitive coordination of nodes and / or networks that may implement their own cognitive control capacity. On this layer the CCE-CCE reference point constitutes as an interface between distributed cognitive decision-making. Typically, the self-growing attribute is realized on this layer, in particular coordinating purposes and life cycles i.e., node and network configuration, topology changes, coexistence and integration, etc.

Administrative: This layer collects cognitive control capacities and interaction between cognitive engines required to coordinate between collections of nodes and / or networks each under their dedicated cognitive control. On this layer the CCE-CCE reference point mainly constitutes between cognitive engines of the operational layer. The interface thus may have a dedicated objective on the exchange of knowledge (e.g. context, rules or policies) enabling decisions on coordinated activities of self-growing networks e.g., incentive based collaboration or integration of self-growing networks. Clearly, more restrictive security requirements apply to interactions between cognitive engines on this layer.

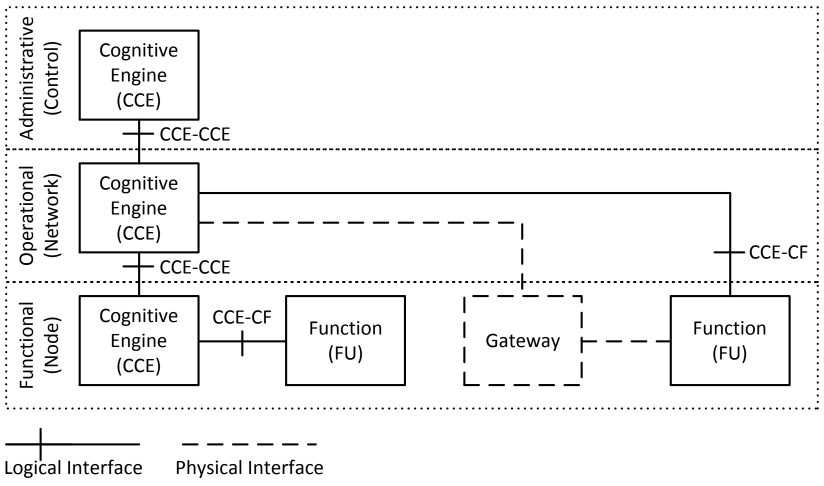

Figure 4-1: Layered logical self-growing architecture.

In a practical implementation boundaries between layers may be fuzzy to some degree since certain topologies or configurations may require cognitive engines to coalesce across layers. Figure 4-1 here provides an example for a coordinating cognitive engine also controlling node level functionality due to a lack of cognitive control features on this layer. In this case the reference point CCE-CF between cognitive engine and function is cross-layer and the CCE-CCE interface is not implemented in this direction.

As an example, low-profile wireless sensor networks (WSNs) may be considered that allow configuration of sensor nodes via an access gateway but do not implement self-configuration or cognitive capacity on their own. In order to integrate in a self-growing manner, some remote entity then needs to implement cognitive control functions required.

From a perspective of potential migration paths towards a self-growing architecture Figure 4-2 defines three basic configurations:

A full self-growing architecture consists of reconfigurable nodes and / or networks in that nodes and / or networks associate with (potentially collocated) cognitive decision-making capacity, which in turn controls collocated or distributed functions (implementing reconfiguration capacity) via the CCE-CF interface. Distributed cognitive engines are communicating and coordinating utilizing the CCE-CCE interface to control the self-growing capacity of the compound system. In addition, a focussed CCE-CCE interface is realized to enable coexistence, coordination and collaboration between self-growing systems, potentially exchanging information required to prepare and initiate merging of these self-growing systems into a single system.

A full self-growing architecture enables planning and decision-making of purposes and life cycles in 'it's own network', can coordinate with neighbouring networks, and can evaluate and judge upon potential benefits of cooperating or joining with neighbouring networks. In addition it may detect potential conflicts in purposes or life cycles prior to initiate collaboration between networks (see UC-02, UC-08, UC-09, UC-15 and also UC-03, UC-10, UC-13, UC-14, UC-16 [2], [3]).

A full-self-growing architecture is not yet available as a practical realization.

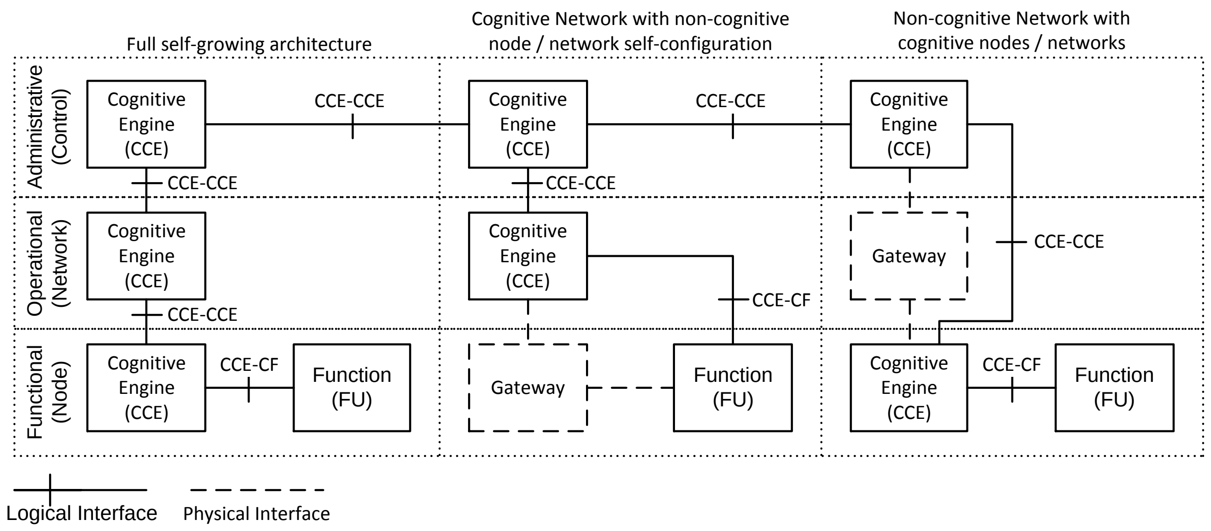
Full self-growing architecture
Cognitive Network with non-cognitive node / network self-configuration
Non-cognitive Network with cognitive nodes / networks
Administrative (Control)
Operational (Network)
Functional (Node)
Cognitive Engine (CCE)
CCE-CCE
CCE-CF
Function (FU)
Gateway
Logical Interface
Physical Interface

Figure 4-2: Basic applications of a self-growing architecture to various topologies.

A cognitive network consisting of nodes and / or networks without dedicated self-x or cognitive capacity consists of nodes and / or networks realizing a certain degree of configurability controlled, for example, by implementing a control and configuration interface per node or by some dedicated gateway entity responsible for configuring nodes and network (which in turn might be collocated to a node). This configuration capacity is assumed to be not of cognitive nature i.e., algorithmic or heuristic. In this case a coordinating cognitive engine, mainly responsible for implementing the self-growing attribute, may also take responsibility for configuring nodes and / or networks accordingly. Hence, the operational layer constitutes both the CCE-CCE and the CCE-CF reference points to allow cognitive engines associated with this layer to communicate with each other (e.g. in a distributed realisation) and with functions throughout the network that must be controlled and configured. Cognitive engines must rely in this on gateway functions either providing access to node and / or network configuration functions, representing this functionality as a proxy, or as a controller or service gateway with some additional abstraction of the functions controlled. As a proxy it may also represent the configuration capacity of a collection of nodes and / or networks. Next layer cognitive functions are considered equivalent to a full self-growing architecture as discussed earlier in this section.

This architecture enables planning and decision-making on purposes and life cycles in 'it's own network' comparable to a full self-growing architecture. Yet it is restricted in the set of life cycles and purposes it can attain by the configuration capacity reflected by the gateway, which might be significantly less than provided by single nodes, since individual policies may apply potentially prohibiting certain configurations.

This architectural model applies to a number of options based on existing network architectures. For example, a WSN consisting of low-profile nodes may support control of sensor acquisition rate, communication frequency and communication routes via a gateway providing an API or a Web interface. Usually optimized for increased battery lifetime a coordinating cognitive engine can use these functions for reconfiguring the WSN to increase sensor acquisition rate and to reduce communication delay by route optimization at the same time. For the cost of increased battery drainage of some WSN nodes, collaborating nodes may gain knowledge required to attain the targeted purpose e.g., to determine the area impacted by an incident and providing communication services for this area (see UC-01, UC-05, UC-06, UC-07, UC-11 [2], [3]).

A network consisting of cognitive nodes and / or networks without per-network cognitive coordination of self-growing capacity. This architecture consists of reconfigurable nodes and / or networks in that nodes and / or networks associate with (potentially collocated) cognitive decision-making capacity, which in turn controls collocated or distributed functions (implementing reconfiguration capacity) via the CCE-CF interface. Distributed cognitive engines are communicating and coordinating utilizing the CCE-CCE. Since this architecture lacks a cognitive control of the self-growing capacity across a collection of nodes and networks (on the operational level) it rather constitutes a loose collection of coexisting systems that can be coordinated in a self-growing manner (on the administrative level) but lack the inherent support (and knowledge) for attaining full self-growing capacity.

This architecture thus enables planning and decision-making on purposes and life cycles across networks but doesn't provide full functionality when coordinating its cognitive nodes / networks in a self-growing manner. That is, coordination across networks cannot make full use of the self-configurability of participating nodes / networks although these nodes are collaborative in their cognitive control. This is based on the assumption that a fully decentralized and collaborative decision making scheme a) cannot achieve the same or better performance than a partly centralized decision making architecture, and b) cannot control neighbouring nodes / networks remotely that do not provide local cognitive control capacity.

This architectural model applies to a number of options migrating existing networks into self-growing networks. For example, deploying cognitive nodes into an existing network and deploying cognitive control to this network can be managed independently, increasing heterogeneity in the first place and increasing complexity of cognitive control on the operational layer. This evolution of cognitive control across layers towards a full self-growing architecture should avoid the need for 'synchronising' complex intermediate steps. Hence, a network could be partitioned (regarding its cognitive capacity) by creating dedicated sub-networks and collecting self-growing nodes / networks and functional (non-cognitive) nodes logically into separate domains. The coordination is then maintained on the administrative layer in the same way as for heterogeneous networks, one of these is self-growing while the others are not (see UC-04, UC-07, UC-11, UC-12 [2], [3]).

Table 4-1 summarizes the applicability of the self-growing architecture to the use cases considered in [2] and [3] with respect to the layered approach presented here, focussing on the requirements the use case places on the presence of cognitive decision making capacity within the various layers.

Use case	**Requires architecture enabling**				
	Full self-growing (1)	**Node configu-rability (2)**	**Node self-growing (functional)**	**Network self-growing (operational)**	**Inter-network self-growing (administrative)**
UC-01: Energy Optimization in a moving vehicle with capacity and coverage limits	No, opera-tional	Partially (relays, APs or mobile routers only)	No	Coexistence (3) (under energy constraints), heterogeneity (6)	WiMAX BTS coordination (for efficiency under coverage constraints)
UC-02: Energy Optimization in an Office environment under coverage constraints	Yes	All Nodes	Association(5)	Coexistence (3), Scalability (4), (under energy and coverage constraints)	WiFi AP, 3G Femto coordination (for efficiency under coverage constraints)
UC-03: Energy Optimization for Self-Growing Office environment under coverage and capacity constraints	No, functional, operational	WSN gateways	Association(5) (with WiFI AP or Femto)	Scalability (4) (under energy constraints), Gateway, AP, Femto coordination	No (except if different operators)
UC-04: Network reconfiguration following the introduction of new nodes	No, functional, administra-tive	Partially (multi-mode nodes only)	Association(5) (with AP or Femto/BTS)	Scalability (4), AP, Femto and BTS coordination, heterogeneity (6)	Coordinating heterogeneity (6) (between self-growing and conventional)
UC-05: Switch on-off of nodes for Energy Efficiency in Heterogeneous Networks	No, opera-tional	Partially (nodes may be used as sensors)	No	Scalability (4) (under energy constraints)	No (except if different operators)
UC-06: Cooperative DAS nodes configuration	No, operational, administra-tive	Partially (nodes may need to adapt transmission range and direction)	Optimization (7) (under energy and coverage constraints)	BTS and MT coordination	BTS coordination (for efficiency under coverage constraints)

UC-07: Cooperative relay for Energy Efficiency	No, functional or operational, administrative	Partially (relay nodes only)	Optimization (7) (under energy, coverage and load constraints)	BTS and relay node coordination, coordination between relays	BTS coordination (under relay node topology and coverage constraints)
UC-08: Energy-aware end-to-end delay optimization	Yes	Partially (WSN gateways, APs, BTSs, multi-mode nodes)	Detection (8), association (5), communica-tion (with neighbouring nodes, APs etc.)	Coordination between self-growing nodes/ networks, control of nodes/ networks, integration of functional capacities	Coordination and decision-making prior to association (5)
UC-09: Purpose-driven network reconfiguration during an emergency situation	Yes	Partially (WSN gateways, APs, BTSs, multi-mode nodes)	Detection (8), association (5), communica-tion (with neighbouring nodes, APs etc.), interface creation (9)	Coordination and modification of purposes and life cycles, enabling priority access (for authorities)	Coordination and decision-making prior to association (5), enabling potentially destructive decisions
UC-10: Cognitive Coexistence and self growing for white space operation	No, functional, operational	Partially (multi-mode nodes)	Optimization (7), coexistence (3) (under spectrum use efficiency and interference constraints)	Scalability (4) (under efficiency and interference constraints), coexistence (3) (under regulatory constraints)	No (except if different operators)
UC-11: Energy optimization of co-located wireless networks in a home/office environment	No, functional or operational, administra-tive	Partially (multi-mode nodes only, nodes may be used as sensors)	Optimization (7) (under energy and coverage constraints), coexistence (3)	Scalability (4) (under energy constraints), AP, Femto and BTS coordination, heterogeneity (6)	Coordination and decision-making prior to association (5), coordination between operators domains
UC-12: Self-adaptation of a reconfigurable wireless terminal	No, functional	Yes	Optimization (7) (goals remotely set)	Scalability (4), coordination between nodes/ networks	No
UC-13: Home	No, functional,	Partially (WSN	Association (5)	Coordination	No (except if

Monitoring Energy Optimization	operational	gateways, configurable sensors)	(with available communica-tion infra-structure), coexistence [(3)]	and modification of purposes and life cycles	different operators)
UC-14: Cooperation Enablers in Home Gateway Environments	No, functional, operational	Yes	Optimization [(7)], coexistence [(3)], association [(5)]	Scalability [(4)], coordination between nodes/net-works	No (except if different operators)
UC-15: Dynamic Meeting Setup Flexible Office/Building Environments	Yes	Partially (multi-mode nodes)	Optimization [(7)], coexistence [(3)], association [(5)] (with available communi-cation infra-structure),	Scalability [(4)], coordination between nodes/ networks, control of purposes and life cycles	Coordination and decision-making prior to association [(5)], coordination between operators domains
UC-16: Collaboration of Different Technologies	No, functional, operational	Partially (multi-mode nodes, WSN gateways)	Detection [(8)], association [(5)], communi-cation	Coordination and modification of purposes and life cycles, enabling priority access (for disruptive use)	No (except if different operators)

(1)	No, if the objectives of a use case can be achieved (partly) without mandatory self-growing cognitive control. Yes, if cognitive control is required on all layers (functional, operational and administrative). Functional, Operational or Administrative, if the corresponding layer of cognitive control is required.
(2)	Conventional configurability based on, for example, existing network management interfaces.
(3)	In terms of coexistence between a reconfigured node and other (managed) nodes or between different technologies, otherwise 'scalability' applies.
(4)	In terms of number of nodes addressing optimization of RF emission, node coexistence, off-loading, interference control etc.
(5)	Nodes or networks may also decide autonomously (uncoordinated) to join or dissolve.
(6)	Coordination may be required between configurable and non-configurable nodes.
(7)	In terms of optimizing autonomously / collaborative (not self-growing) a limited set of operating parameters to a goal set by node / network-external controls.
(8)	In terms of detecting the presence of neighbouring nodes of a different technology and being able to obtain information needed to establish an association with these nodes by negotiating on a common method for communication.

(9)	In terms of making new functions available through reconfiguration e.g., functions to override embedded policies by administrative means.

Table 4-1: Applicability of layered self-growing architecture objectives to use cases.

Table 4-1 makes some assumptions on the realisation of use cases regarding their requirements for implementing cognitive control on a certain layer:

- If a use case is assumed to rely on the presence of cognitive nodes (e.g. cognitive radio, cognitive decision making is done with a clear focus on configuring a collocated device or collection of devices), it is said to require cognitive control on the functional level,
- If a use case is assumed to rely on configurable nodes but there is no clear requirement for node-local cognition (i.e. the device is configured remotely), it is said that no cognitive control on the functional level is needed,
- If a use case is assumed to rely on a network-wide cognitive control, does not realise node-local decision making, or likely relies on coordination or collaboration among node-local cognitive engines, it is said to require cognitive control on the operational level,
- If a use case is assumed to rely on device-local cognition but there is no clear requirement for cognitive decision making across a multitude of nodes or it is likely that a fully decentralised cognitive control (i.e. without needing centralized elements such as common rule base, policy store or ontology) is feasible, it is said that no cognitive control on the operational level is needed,
- If a use case is assumed to rely on a cross-network cognitive control, or likely relies on coordination or collaboration among network-wide cognitive engines, it is said to require cognitive control on the administrative level,
- If a use case includes collections of (heterogeneous) nodes that may have either configuration capacity (without cognitive control) or associate with a cognitive control on the functional or operational level, but this cognitive control does not apply to all nodes, it is also said to require cognitive control on the administrative level.

Multiple realisation options exist for all use cases and no clear distinction can be made in some cases. Thus, Table 4-1 also tries to mark use cases that likely can be realised by both, for example, network-wide control by a dedicated cognitive engine or, alternatively, by decentralized and tightly coupled node-local (functional) cognitive control.

From the discussion above a stratum model can be developed resembling that given earlier for UMTS [4]. The purpose of this model here is to depict the interaction (in terms of information exchange across interfaces) of cognitive engines in the CONSERN architecture. As shown in Figure 4-3 the role of network nodes in the CONSERN architecture is conventional but slightly more complex due to their configurability and other self-x capacities. For the stratum model it is assumed that cognitive decision-making is incorporated either as a cognitive function with a network node (potentially one or more of those shown in Figure 4-3) or with a dedicated node hosting a decision engine e.g., as a dedicated service node.

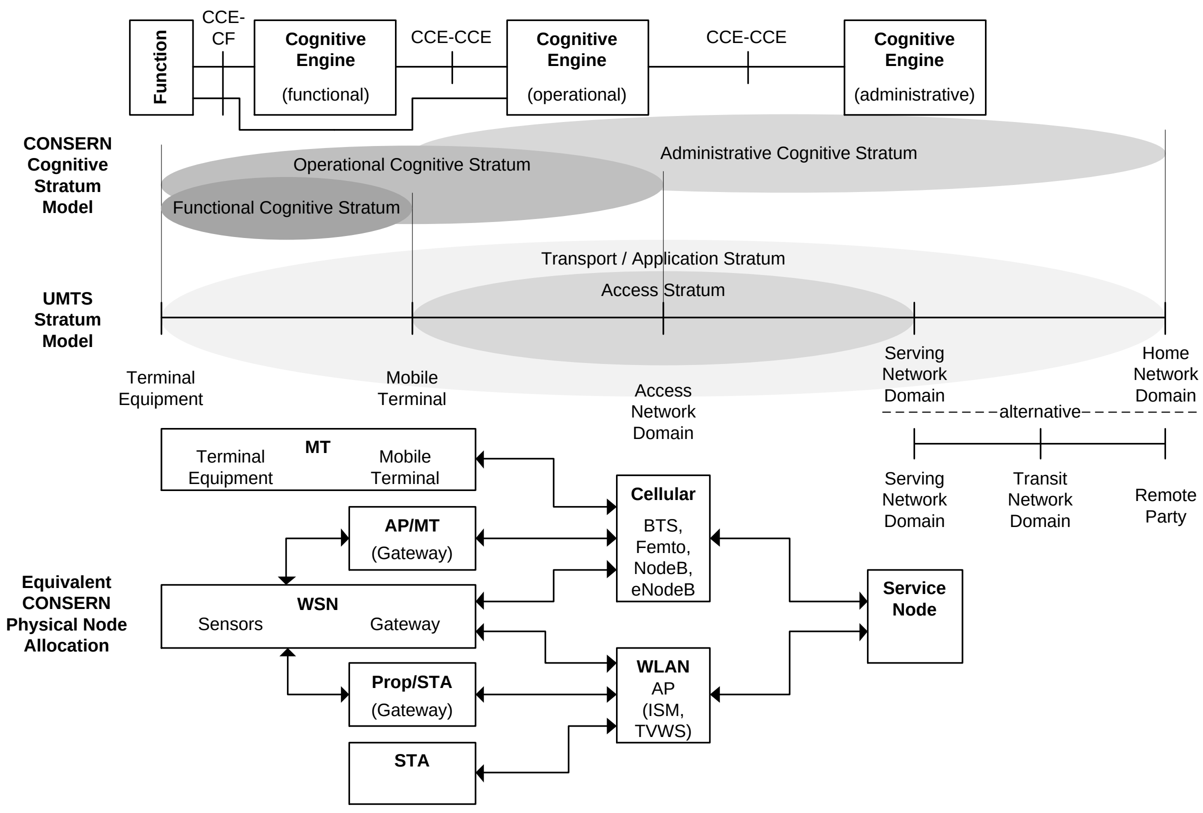

Figure 4-3: Stratum Model of Cognitive Engines realizing Self-Growing.

Hence, the functional layer shown in Figure 4-1 and Figure 4-2 is associated with the functional cognitive stratum shown in Figure 4-3. Clearly, the corresponding association also exists for the operational and administrative layer. A mapping of logical interfaces CCE-CF and CCE-CCE and the physical interfaces then results from this association. In detail, the following assumptions apply to the cognitive strata:

The **functional cognitive stratum** supports communication between decision engines and functions to configure, and between cognitive engines for decisions on configurations. It includes consideration of the transmission format of this configuration information as well as related control, status and measurement information. The logical interfaces CCE-CF and CCE-CCE are realized by mechanisms that map to internal interfaces (e.g. APIs) or to standardized communication means e.g., TCP/IP, UDP/IP, CORBA IDL, OMG SCA.

The **operational cognitive stratum** supports communication between decision engines and functions to configure, and between cognitive engines for decisions on configurations. It also supports the communication of 'knowledge' in terms of policies, rule bases, resource description etc. to realise distributed decision making. The operational cognitive stratum includes consideration of the transmission format of this information as well as related control, status and measurement information. The logical interfaces CCE-CF and CCE-CCE are realized by mechanisms that map to standardized communication protocols.

The **administrative cognitive stratum** supports communication between cognitive engines for decisions on configurations, decisions on collaboration of remote operational cognitive engines, and for conflict resolution. It also supports the communication of 'knowledge' in terms of policies, rule bases, resource description etc. to realise distributed decision making and synchronisation in actions and knowledge between operational cognitive engines. The administrative cognitive stratum includes consideration of the transmission format of this information as well as related control, status and measurement information. The logical interface CCE-CCE is realized by mechanisms that map to standardized communication protocols satisfying security and reliability requirements appropriate for the communication of sensitive information across unreliable transit networks.

5. CONSERN Architecture

In this section, the CONSERN architecture is presented. The architecture is the realization of the architectural model presented in section 4 and is designed to fulfil the requirements presented in section 2.

A CONSERN network comprises one or more CONSERN Entities (CEs) containing the Communication Services (COM) function and may contain the modules CONSERN Cognitive Engine (CCE) and Translation function (TRA). The CCE as earlier described in section 3.1.2 is unique to the CONSERN architecture and is mandatory to include in at least one of the CEs. The COM is mandatory to all CEs in that it provides the communication to other CEs or FUs while the TRA is only mandatory to the CEs controlling a FU. The Managed Resource models a physical network node as presented in section 6.3. The interconnection of a CONSERN enabled network (network nodes are enabled with CEs) with legacy systems is realized through the CONSERN Configurable Gateway (CCG) (see section 6.6). CONSERN Sensor Coordinator (CSC) and Sensor Data Aggregator are FUs providing the coordination and control of the existing sensor networks. Finally, the CONSERN Policy Manager (CPM) is introduced to translate high-level business goals injected by the Policy Provider to specific policy rules that will be enforced to the network elements (see section 0).

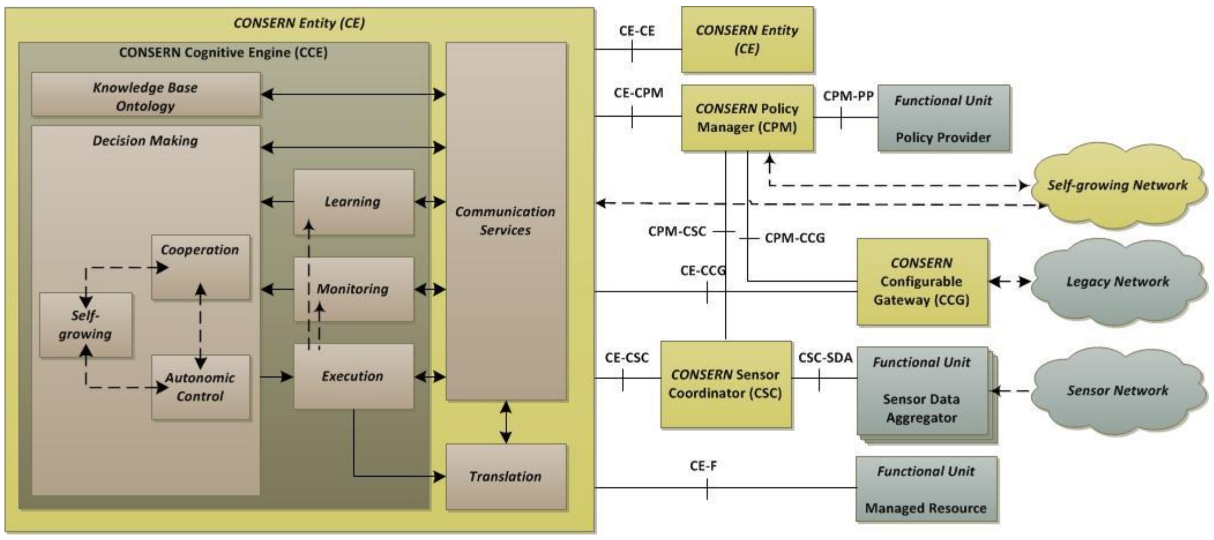

Figure 5-1: The CONSERN Architecture– Type A CE.

As depicted in the previous figure, the Decision Making function of a CE is composed of the three functions: Self-growing, Cooperation, and Autonomic Control. Different types of CCEs exist based on the existence or not of these functions and are the following:

Type A: A CE that incorporates all of the capabilities designed in the scope of CONSERN i.e., Self-growing, Cooperation, and Autonomic Control. The CE presented in Figure 5-1 is of Type A.

Type B: A CE that is enabled with Cooperation and Autonomic Control, but does not include Self-growing functionality. These types of nodes are cognitive nodes but they cannot drive the self-growing paradigm. An example of such architecture is presented in Figure 5-2.

Type C: In this category belong the nodes that incorporate only the Autonomic Control. They are capable of performing only basic and localized management operations. They can be controlled by nodes that provide higher level of intelligence i.e., Type A and Type B enabled nodes. An example of Type C architecture is depicted in Figure 5-3.

Type D: A CE that is enabled with Self-growing and the Autonomic Control, but does not support Cooperation is presented in Figure 5-4.

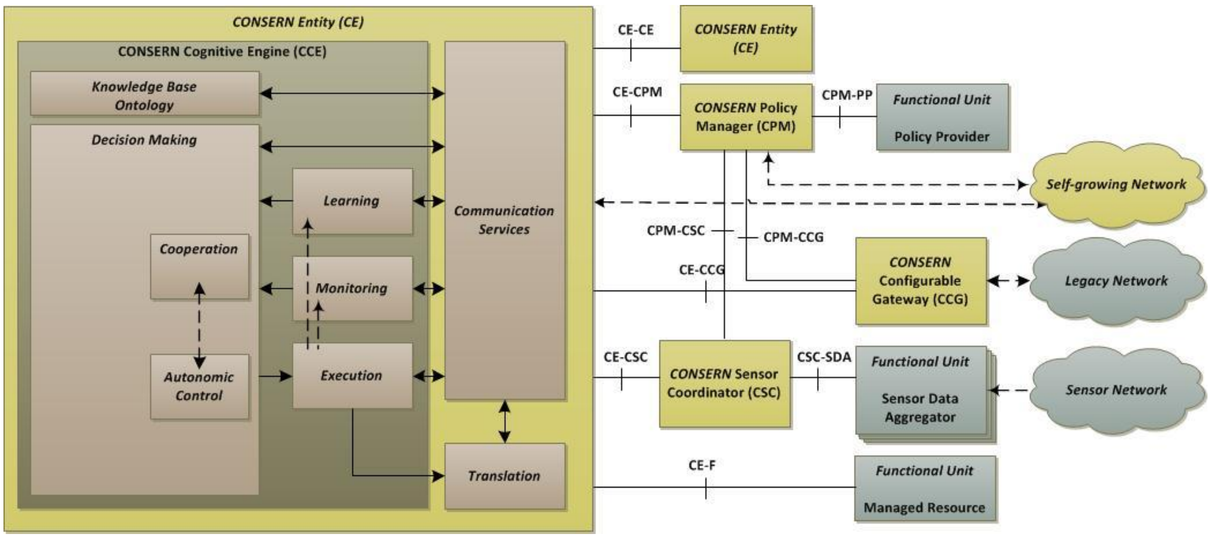

Figure 5-2: The CONSERN Architecture – Type B CE.

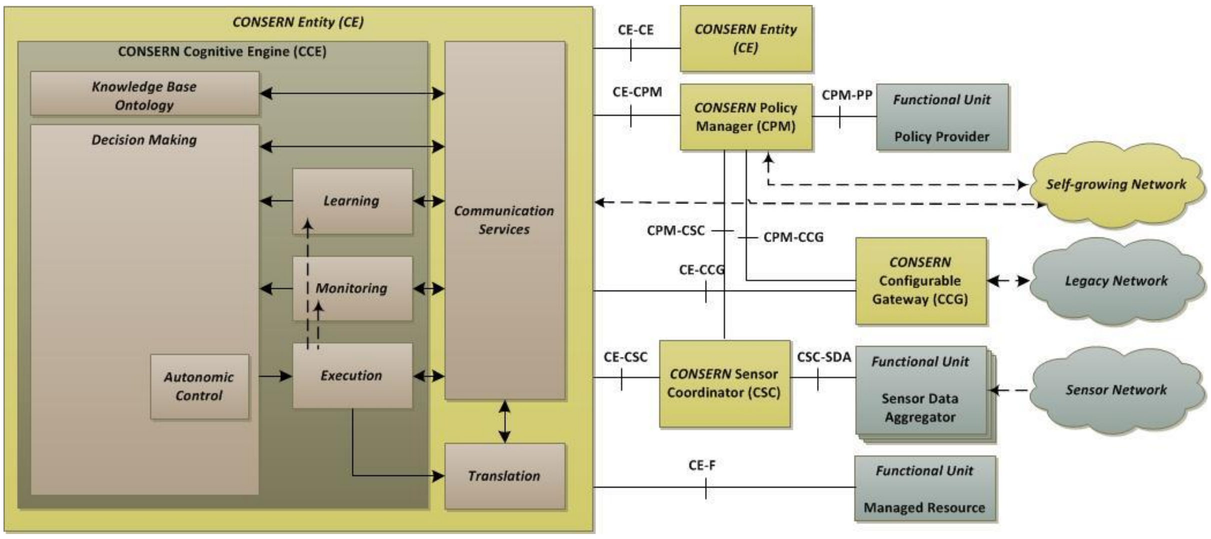

Figure 5-3: The CONSERN Architecture – Type C CE.

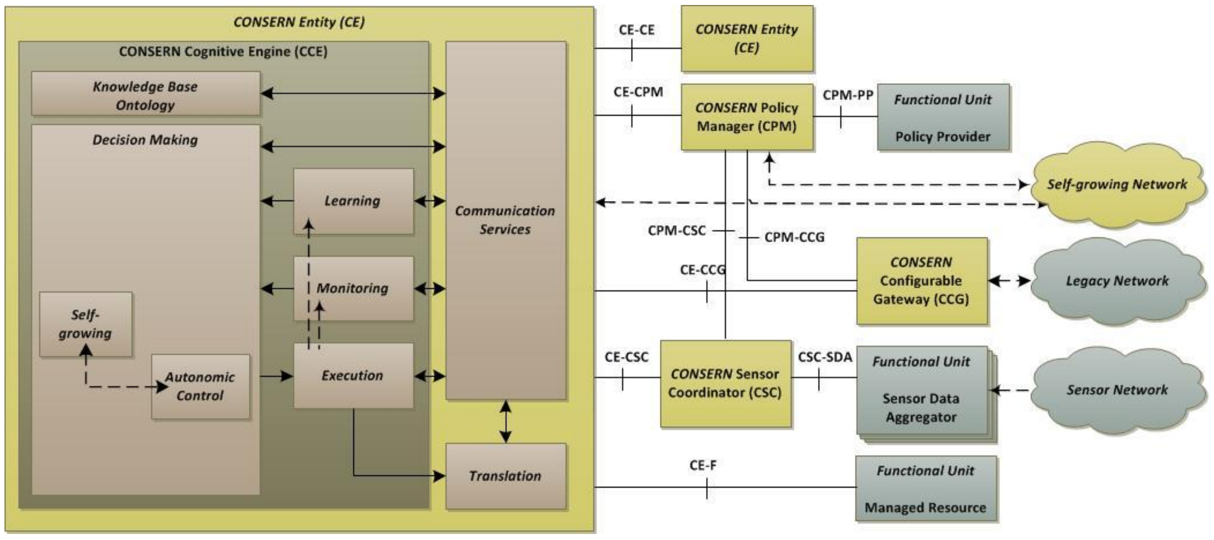

Figure 5-4: CONSERN Architecture – Type D CE.

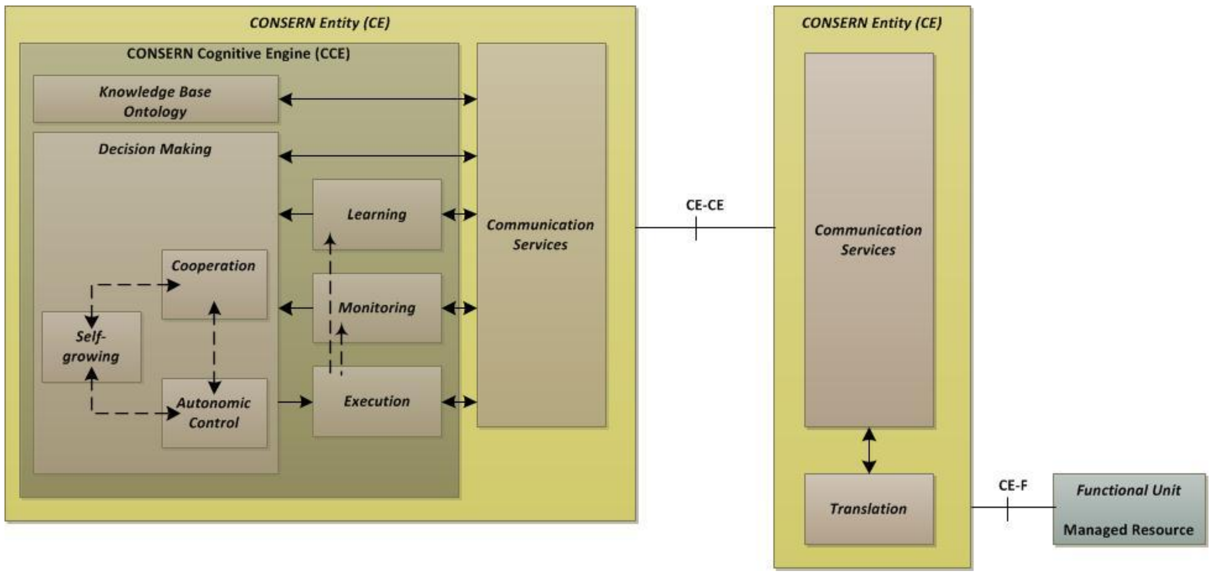

Figure 5-5: CE - distributed approach.

As mentioned above, the CE is designed with the endogenous capability to operate fully distributed in the sense that not all of its components need to be present in a single CE. Figure 5-5 presents such a possible configuration in which the TRA is moved to a CE controlling a FU but not containing a CCE.

In the following section, we present a detailed description for the mapping of the self-growing enablers to the proposed CONSERN Architecture.

5.1 From Self-growing Enablers to the CONSERN Architecture

This section performs a mapping between the functions enabling Self-growing and the CONSERN architecture. The two approaches can be considered as orthogonal to each other: functions enabling self-growing are – mostly – objective specific thus serving a well-defined objective around self-growing scenarios. On the other hand, the CONSERN architecture needs to support mechanisms and functionalities serving the other objectives of the project, cooperative management and control and energy optimisation. Clearly, this is not a deterioration regarding the scope of the self-growing architecture as self-growing has been identified as a novel paradigm enabling advanced energy efficiency mechanisms.

5.1.1 *Modules and Functions enabling self-growing – a UML model*

Figure 5-6 provides a formal overview of the functions provided by the Functional Unit (FU). The FU (stereotyped as Class of Module) enables Self-growing (stereotyped as Attribute) which is realised by the CCE which controls the execution of the FU's functions as detailed by the classes InitiateExecution and ControlExecution.

A set of Functionality Groups (stereotype) has been included in the model reflecting key functional aspects which are required for Self-growing operation; those Functional Groups are provided by the FU class of module. Each Functional Group is composed by a set of functions; where possible, a set of indicative attributes and operations have been identified to better outline those function's scope and managed resources.

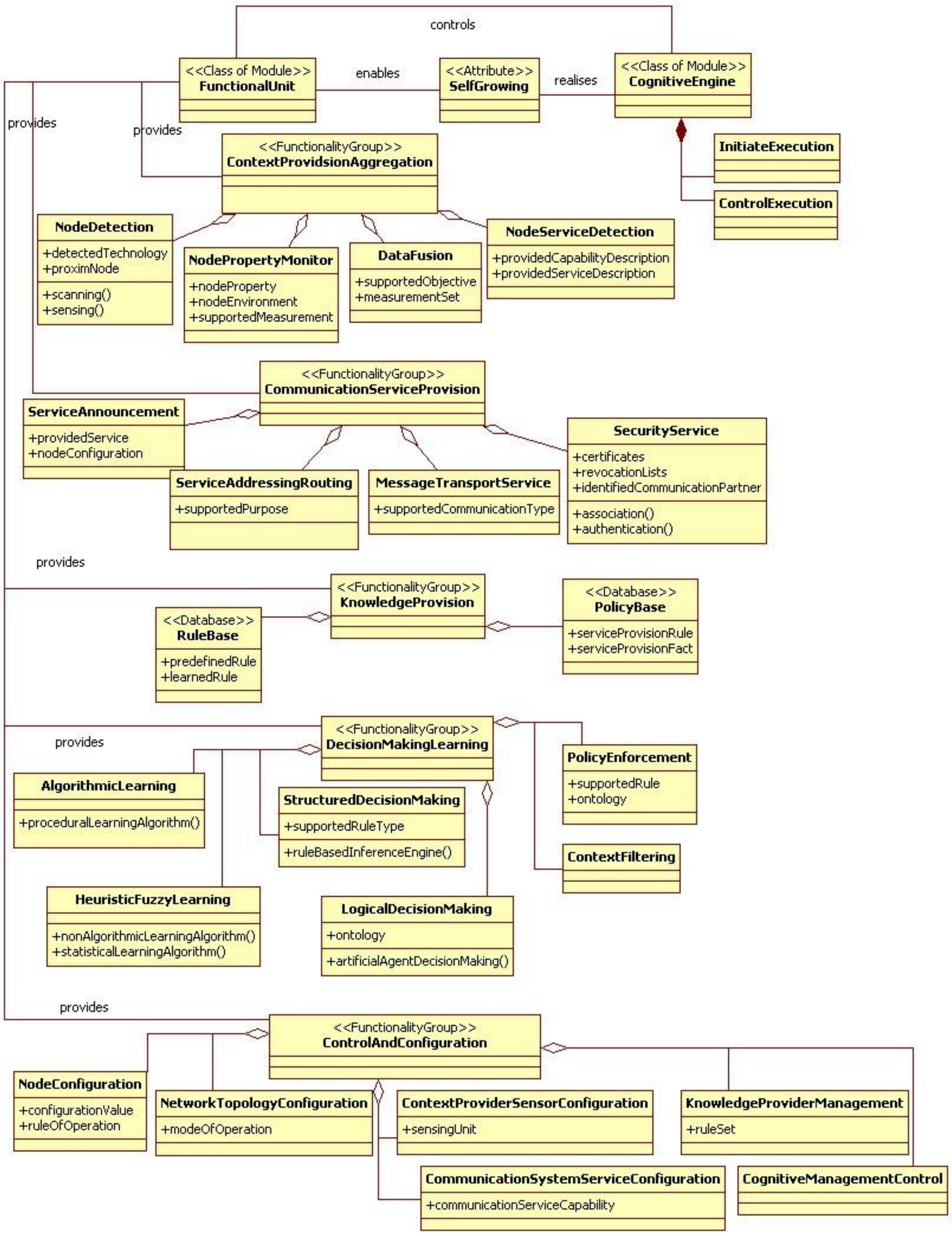

Figure 5-6: FU functions - UML Model.

5.1.2 *Mapping analysis*

This section briefly presents the assumptions and the procedure applicable for the mapping.

- Assumptions for the mapping

  - The mapping will be performed at the operational level where network nodes and networks are considered as a collection of functions,
  - The mapping will be based on the functions, services and capabilities which are provided by the FU module; CCE module initiates and controls the execution of those functions,
  - Functional Entities in the CONSERN architecture represent a set of utility functions which can support multiple objectives; at a second level of detail, Functional Entities will capture,
    - Objective-specific functions as described in the Self-growing FU module,
    - Objective-specific functions reflecting energy awareness / efficiency as being enabled / facilitated by Self-growing specific; such functions can be provided by WP2 and WP3 at a later stage,
  - The mapping was based on the assumption that the scope of functions is the realization of the self-growing paradigm. Thus, the mapping can be further elaborated to incorporate functions specified for cooperation.
- Mapping procedure
  - At a first step it is useful to clarify the scope and the applicability of each of the functions described within Self-Growing FU:
    - Functions servicing only self-growing procedures and scenarios,
    - Functions that are envisaged to be applied to a broader set of application/attributes.

5.1.3 *Mapping specification*

Table 5-1 presents the specified mapping. The first column presents the FU's functions and the second a short description of each function's scope and the proposed Functional Entity or generic function for mapping of the CONSERN architecture.

FU Function	Scope – Mapping to Functional Entity/Generic function
NodeDetection	Detection of other technologies or specific nodes for integration on self-growing basis. Cooperative decision making may follow. ➔ Monitoring function
NodePropertyMonitor	Monitoring at node level ➔ Monitoring function ➔ Communication Services function (reporting)
DataFusion	Applied to measurements according to a given objective. ➔ Sensor Data Aggregator ➔ Further processing according to the objective. For example data enabling cooperative power control → Monitoring

FU Function	Scope – Mapping to Functional Entity/Generic function
	function
NodeServiceDetection	Detection of services and capabilities of other nodes following node detection. ➔ Communication Services function ➔ Monitoring function
NodeServiceCapabilityDescription	Capability to inform other entities of the services and capabilities and node may provide. ➔ Knowledge Base Ontology
ServiceAnnouncement	Single node advertises its supported services to the network ➔ Decision Making function – Autonomic control function ➔ Communication Services function (dissemination)
ServiceAddressingRouting	Connection of two networks (different purposes) ➔ Execution function ➔ Learning function Address and reach nodes to the other network ➔ Communication Services function ➔ CONSERN Configurable Gateway (communication with legacy nodes/networks)
MessageTransportService	Send-receive message ➔ Communication Services ➔ CONSERN Configurable Gateway (communication with legacy nodes/networks)
SecurityService	Association, authentication etc ➔ Communication Services function ➔ CONSERN Configurable Gateway (communication with legacy nodes/networks)
RuleBase	Rule sets needed for the decision making ➔ CONSERN Policy Manager ➔ Knowledge Base Ontology (local rules etc.)
PolicyBase (Policy Provider)	➔ CONSERN Policy Manager

FU Function	Scope – Mapping to Functional Entity/Generic function
DecisionMaking	➔ Decision Making function (the different features (logical, structural, etc) should be included when progressed in the context of T4.3)
Learning	➔ Learning function (the different features (fuzzy, etc) should be included when progressed in the context of T4.3)
PolicyEnforcement	Involves rules and ontology for decision making ➔ Execution function
ContextFiltering	Processing of contextual data acquired through monitoring or interaction with other CONSERN nodes ➔ Monitoring function
NodeConfiguration	Execution of the configuration actions ➔ Execution function ➔ Translation function (translation of execution commands to vendor/hardware/vendor specific actions)
NetworkTopologyConfiguration	Network topology reorganization because of planning optimization, self-growing paradigm realization ➔ Execution function ➔ Translation function (translation of execution commands to vendor/hardware/vendor specific actions) ➔ Communication Services function (disseminate configuration actions to other nodes)
ContextProviderSensorConfiguration	Configuration of sensing units based on the decisions of the “controlling” CONSERN Entities ➔ Communication Services function ➔ CONSERN Sensor Coordinator
CommunicationSystemServiceConfiguration	Configuration of communication service capabilities i.e., handshaking procedures among CONSERN nodes to decide upon communication parameters and capabilities ➔ Communication Services function
KnowledgeProviderManagement	Communication with the knowledge provider management service

FU Function	Scope – Mapping to Functional Entity/Generic function
	➔ CONSERN Policy Manager (global view policies) ➔ Knowledge Base Ontology (storage of policies locally on the CONSERN Entities)
CognitiveManagementControl	➔ Communication Services function ➔ CONSERN Configurable Gateway

Table 5-1: Mapping of functions to functional entities.

6. Module Specifications

In this section, a high level specification of the modules that compose the CONSERN architecture is presented. The proposed modules enable a network to operate in a self-growing, self-organizing, and energy efficient manner. In the next stage of the architecture design, these modules will be revised, if necessary, and they will be further decoupled into atomic functions based on the mechanisms that will be designed and implemented in the context of the project.

6.1 CONSERN Entity

The CONSERN Entity (CE) enables the self-growing paradigm, autonomicity, and cooperation of the network nodes. A complete CE is composed of three main functional components: the CCE, the Communication Services (COM), and the Translation (TRA). The components of a CE may be distributed between several CEs and the TRA is only needed in CEs controlling a FU. An example of this architecture can be seen in Figure 5-5. The components of a CE and their internal structure are presented in the following sections.

6.1.1 *CONSERN Cognitive Engine*

The CONSERN Cognitive Engine (CCE) provides all the intelligence so as the CE to realize the self-growing paradigm and the cooperation and energy-efficient mechanisms developed in the context of the project. The CCE is not an “atomic” function but it is composed by lower layer functions each of which is responsible to execute specific parts of the cognition loop. In the following sections, a detailed analysis of those functions is presented.

6.1.1.1 Knowledge Base Ontology

The Knowledge Base Ontology (KBO) contains information about the state of the network. The information stored in KBO can be classified into five logical groups[4] as depicted in Figure 6-1.

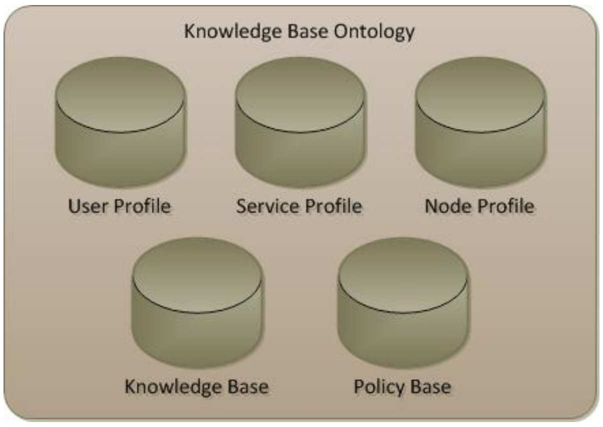

Figure 6-1: Knowledge Base Ontology internal structure – abstract view.

User Profile: contains information related to users, such as personal information, subscriptions, preferences, etc.

Service Profile: contains information about the status and requirements of active services, such as parties involved, required resources, billing, etc.

Node Profile: contains information about the static and dynamic profile of the node. Static profile includes the hardware specifications of the node, initial setup etc. Dynamic profile includes current

[4] A similar approach has been proposed in ASA [9].

resource utilization, state of the node, etc. The node profile also contains information on all possible configuration alternatives of a node e.g., SDR functionality in the node enables the node to potentially switch from one technology to another.

Policy Base: high-level policies specified by the operator in the CPM are elaborated to explicitly match the operation of the network nodes. These node specific policy rules are stored in the Policy Base.

Knowledge Base: contains information about events (triggers) and actions. Knowledge is initially provided by the operator and is thereinafter built through the learning procedure that is executed at various time scales e.g., each time an event is triggered and an action is performed. Several tools can be used for knowledge building such as statistical analysis, etc.

The use of the Knowledge Base is to enable a node to perform a certain action upon the triggering of a certain event without the need to execute the control loops in the CCE. This reduces the processing usage in the nodes and increases the intelligence level in the network.

The internal structure of the Knowledge Base is presented in Figure 6-2. Knowledge Base is composed of three layers:

- **Self-growing knowledge:** Knowledge related with the self-growing algorithms is stored in this layer. Usually it's high-level scope in the sense that it describes joint actions from multiple nodes of the network.
- **Cooperative knowledge:** Information related with actions and events regarding the self-organization of the network. It can also describe joint actions from multiple nodes of the network.
- **Autonomic control knowledge:** It holds specific triggers and actions on node level. This is the absolute least required knowledge in order for a node to be able to operate with some level of autonomicity.

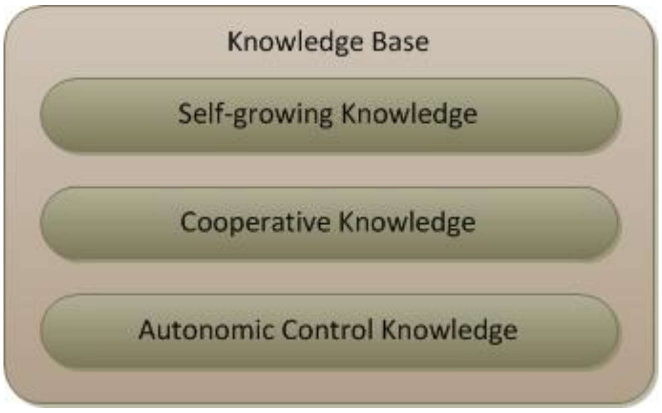

Figure 6-2: Knowledge Base internal structure.

6.1.1.2 Decision Making

The Decision Making function has interface with the Monitoring function to extract the current state of the node and the environment. After the decision on certain actions, Decision Making communicates with the Execution in order to enforce these decisions.

In addition, it is connected with the Learning function in order to exploit existing knowledge and thus minimize the processing overhead of the decision-making procedure.

Finally, an interface between the Decision Making and Communication Services functions is specified so as to enable the communication of the Decision Making function with another CE or a FU.

The Decision Making function is composed of the separate functions that operate in a fully distributed manner: Self-growing (SGN), Cooperation (COP), and Autonomic control (AUC).

6.1.1.2.1 Self-growing

The Self-growing (SGN) function realizes the self-growing paradigm. It holds the state machines of the potential growth of the network towards certain re-configurations. It communicates with the COP and AUC functions to exchange information for decision-making either in the self-growing level or in the self-x level or in the autonomous node level.

Another functionality provided by the SGN is the verification/validation of the configuration actions executed in the past. Specifically, in the occurrence of an event that causes the execution of the CE, specific configuration actions are generated and applied to one or more network elements. Then, an evaluation of the network state should be done in order to verify that these actions were executed correctly and the behaviour of the network is the expected one. SGN provides this functionality. This can be done by evaluating the data collected in KBO after the execution of the SGN.

6.1.1.2.2 Cooperation

The Cooperation (COP) function makes decisions that require cooperation of nodes but without taking into account the self-growing aspects of the network. It acts as “traditional” self-organizing networks. It operates in a higher level than the AUC since it takes more complex decisions. It can also control the operation of the AUC.

6.1.1.2.3 Autonomic Control

The Autonomic Control (AUC) function provides localized decisions in the sense that no information exchange with the environment is needed/performed. Configuration actions affect only the node that currently hosts the CE.

6.1.1.3 Monitoring

The Monitoring function provides information to the Decision Making related with the state of the node and/or the environment. Specifically, it performs the following fundamental operations:

- Dynamic state of the node: Each node has static and dynamic profile. The static profile includes the hardware/software specifications and the initial configuration of the node, while the dynamic profile contains all the runtime information related with the status of the node. Monitoring of the dynamic state of the node is performed by the Monitoring function.
- Situation-awareness: The Monitoring function is responsible with context-acquisition and processing in order to achieve situation-awareness and enable the CE to perform appropriate reconfiguration of the network.

Also, the Monitoring function communicates through Communication Services with the KBO in order to store collected data for further processing and evaluation. Monitoring is connected to the Decision Making as well in order to provide notifications about certain events. For example, Decision Making can configure Monitoring to set specific thresholds for different parameters and when these thresholds are exceeded the Decision Making is notified.

6.1.1.4 Execution

The Execution function receives the decisions from the Decision Making and performs the actual execution of them. It communicates with the Translation in order to enforce the execution commands. It also communicates with the Learning in order to generate knowledge (if possible) and store it to the KBO for future use.

6.1.1.5 Learning

The Learning function enhances the system with learning capabilities. Learning is the process in which the system collects contextual data, policies, and decision-making results generated from the execution of the self-x algorithms in order to build knowledge, which will be used to improve future decision-making and enable the system to operate proactively. Furthermore, it combines and analyses information from the KBO together with newly received information from the Execution function and generates knowledge.

6.1.2 *Communication Services*

The Communication Services (COM) function enables a CE to communicate with FUs or with CEs e.g., CPM, CSG, CSC(s). It also provides bootstrapping and auto-discovery mechanisms. Communication Services and Translation are the minimum functions a network node must implement in order to be considered as a CE, even if it cannot provide the full set of the CE functionality.

6.1.3 *Translation*

The translation (TRA) function provides the translation between abstract configuration commands generated by the CCE into vendor/hardware specific configurations [6]. The Translation should provide the “middleware” that will enable a “legacy” node to become a CONSERN node. From the one side, the Translation communicates with the Execution and Communication Services by using an abstract language (that will be defined in the context of the project), and from the other side it communicates with the physical node using hardware/software/vendor specific languages that will be defined by manufacturers.

6.2 Cognitive loops - processes

The functions that compose the CE operate in two complementary and possibly coordinated cognitive loops/processes, presented in Figure 6-3.

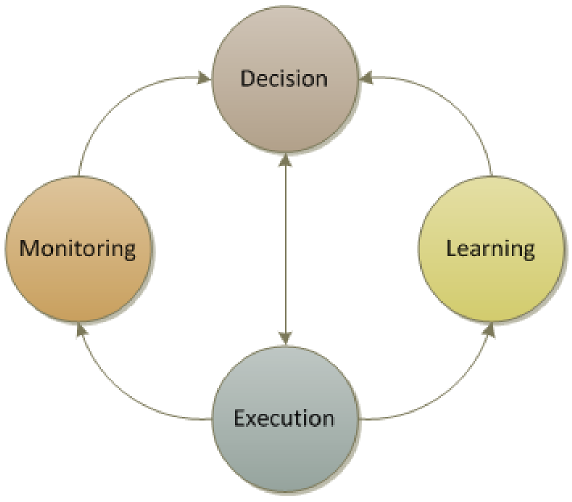

Figure 6-3: Autonomic control loops/processes.

Since the CE is designed to operate in a distributed fashion, not all of the depicted functions need to be physically located in a single CE. Thus, a control loop can be executed between different CEs each of them providing a subset of the functionality of a fully capable CE.

Following this distributed approach, the CONSERN architecture envisages the execution of multiple coordinated cognition loops across different CEs that operate in different time scales. Thus, the network is able to identify and react to events caused by both fast and slow changing network dynamics.

6.2.1 *Self-x loop*

The self-x loop provides autonomicity in the node in the sense of self-configuration, self-optimization, and self-healing without any cognition capabilities. As presented in Figure 6-4, Learning is not part of the loop execution (dark lines).

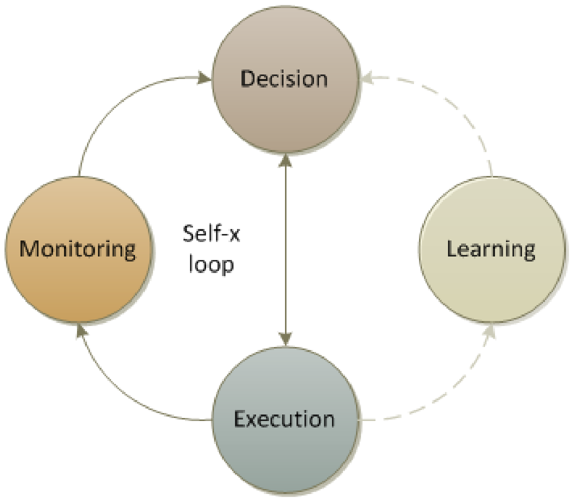

Figure 6-4: Self-x autonomic loop.

6.2.2 *Cognition loop*

The cognition loop enhances the autonomicity of the node by introducing learning in the autonomic loop, see Figure 6-3. This enables the node/network to operate in an advanced way since it can generate knowledge based on actions and events for future use. Thus, the system can operate using prediction mechanisms for future situations and also the processing power used for decision-making is minimized since common events are handled by certain actions already known in the system.

6.3 Managed Resource

The Managed Resource models the vendor specific hardware of a network node. Configuration actions need to be translated into vendor/hardware specific configuration commands and be applied to the respective node [6].

6.4 CONSERN Sensor Coordinator

CONSERN Sensor Coordinator (CSC) is a CE dedicated to aggregate the data from the Sensor Data Aggregators and reporting to the CEs lying in the Network Nodes. Specifically, a CSC reports the measured data to the CE. The CSC does not only aggregate and forward the “raw” data from sensors but already performs pre-processing and analysis. For example, instead of forwarding actual sensor data, statistical information describing the sensed environment could be forwarded. It also receives configuration information from the CE in the case that re-configuration is needed due to specific events triggered, and disseminates the reconfiguration actions to the sensor networks under its control.

6.5 Sensor Data Aggregator

Sensor Data Aggregator collects information from a set of sensors and makes it available to other functional entities. The existence of this entity implies clustering of the sensor nodes. Notice, that the Sensor Data Aggregator is a FU with regards to CONSERN.

6.6 CONSERN Configurable Gateway

The role of the CONSERN Configurable Gateway (CCG) is a CE used to provide an "interface" between CONSERN enabled networks and legacy networks that do not support the functionality specified by the CONSERN. CCG collects all the available contextual data from the legacy systems and reports it to the appropriate CONSERN nodes so as to take the most appropriate decisions. Such a gateway can, for example, provide means to tunnel CONSERN specific protocols via non-CONSERN networks. Also, the CCG should provide means to discover the presence of a CONSERN network/node via the Internet. The gateway is a centralized point used to initiate requests to trigger information from CONSERN entities.

The difference between the CCG and the Translation function is that the Translation function actually translates the execution commands generated by the Execution function of the CE to hardware/software/vendor specific (re-)configuration commands of the network node that the specific CE controls. On the other hand, the CCG provides the interconnection of a CONSERN enabled network with legacy systems so as to be able to acquire contextual data from these networks even if there is no possibility to control their operation.

6.7 CONSERN Policy Manager

The CONSERN Policy Manager (CPM) is a CE that provides the interface with the network administrator (Human-to-Network interface). It also holds the high-level policies that should be applied in the network. Policies are defined by the operator both in the initial setup of the network and during runtime.

The CPM communicates with the KBO through the Communication Services lying in the CE (CE-CPM interface) in order to provide the specific policies related with that node.

Also, the CPM is connected to the CCG via the CPM-CCG interface, as it is very likely that a network operator may want to access the CPM remotely via "legacy" networks.

Finally, the interface CPM-CSC between the CPM and the CSC is defined so as to enable the operator to apply policies directly to the sensor networks (or parts of them).

6.8 Policy Provider

The Policy Provider is a FU in the CONSERN Architecture, which is introduced to enable the operator to define high-level policies that will specify the operation of the network. The Policy Manager can be or not be part of the CPM, thus the interface CPM-PP is specified for the communication.

7. Interface specifications

7.1 Interface categories

Two logical interfaces related to the self-growing capacities addressed by the CONSERN project have been identified so far.

The **CCE-CF logical interface** between a cognitive engine and a functional composition and configuration service is used to obtain information about status and capabilities of a function and to communicate configuration obligations[5].

The **CCE-CCE logical interface** between cognitive engines is used to communicate information required for decision-making. This includes context, rules and policies, configuration capabilities, and configuration obligations.

Complementing interfaces may exist in the context of other self-x capacities of a network out of scope for this document.

In addition, realisations of the logical interfaces CCE-CF and CCE-CCE are composed from multiple function-specific interfaces, allowing a formal specification to consider functional grouping as well as a distinction between remote interfaces (e.g. involving message transport via some communication system) and interfaces between node-collocated entities e.g., an API between an application and a platform implemented service.

Table 7-1 provides an initial attempt to catalogue interfaces required and to categorize them by functional requirements and entities involved. A detailed interface description in the next step will provide a set of primitives, message formats and data structures to exchange, which is left to subsequent sections.

The table should be read as follows:

Interface: the interface implemented (either CCE-CCE or CCE-CF) between CONSERN cognitive engines or between a CONSERN cognitive engine and a function controlled by this engine.

Functional grouping: a logical composition of functions that may result in a common set of interface primitives.

Function class: a logical categorization of functions that consists of primitives required (not outlined in detail here) to realize certain behaviour within a functional group.

Entities involved: building blocks (modules, components or entities) that need to implement functions under consideration.

Interfaces involved: interfaces (either CE-CE and CE-F, or specializations thereof) defining the logical boundary between entities involved with respect to the messages to be exchanged across that boundary that are required to realize the functions considered. As the self-growing specific logical interfaces (CCE-CCE or CCE-CF) can be considered subsets of CE-CE or CE-F, this column denotes the enclosing interfaces affected by the self-growing specific communication under consideration.

As a main result from this attempt to categorize functions vs. interface procedures, it becomes obvious that the implementation of the CCE-CF interface is more complex than the CCE-CCE interface and that the CE-F is a subset of the CE-CE interface. That is due to the fact that a CCE

[5] The term configuration ‘obligation’ herein denotes a ‘rule’ along with a set of parameters describing the configuration target in a rather abstract and platform-independent way. This is more general and robust compared to forwarding configuration parameters, and thus is more suitable for heterogeneous environments.

controlling a CF may not be aware that the FU controlled is ‘behind’ another CE (Figure 5-5) and in consequence CCE-CF primitives must be ‘tunnelled’ through another CE-CE. The ‘tunnel’ requirement also can be seen from Figure 4-1 and Figure 4-2 where the gateway entity reflects this capacity.

Inter-face	Interface description			
	Functional grouping	Function class	Entities involved	Interfaces involved
CCE-CCE	Control & Status	Association	CE (CPM, CCG, CSC), COM	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
		Authentication & trust establishment	CE (CPM, CCG, CSC), COM	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
		Admission control	CE (CPM, CCG, CSC), COM	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
	Configuration	Capability exchange	CE (CPM, CCG, CSC), COM	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
		Configuration primitives	CE (CPM, CCG, CSC), COM	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
	Communication	Establishment & control	COM	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
		Addressing & routing	COM	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
		Message transport	COM	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
		Service announcement & detection	COM	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
	Context	Provider configuration	CE (CPM, CCG, CSC)	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
		Context acquisition & change notification	CE (CPM, CCG, CSC)	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC) CE-F (CPM-PP, CSC-SDA)
		Consumer	CE (CPM, CCG,	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-

Inter-face	Interface description			
	Functional grouping	**Function class**	**Entities involved**	**Interfaces involved**
		configuration	CSC)	CSC)
	Cognition control	Knowledge exchange	CE (CPM, CCG, CSC)	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
		Negotiation	CE (CPM, CCG, CSC)	CE-CE (CE-CPM, CPM-CSC, CPM-CCG, CE-CSC)
		Policy exchange & enforcement	CE (CPM, CCG, CSC)	CE-CE (CE-CPM, CPM-CSC, CPM-CCG)
CCE-CF	Control & Status	Association	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
		Authentication & trust establishment	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
		Admission control	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
	Configuration	Capability exchange	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
		Configuration primitives	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
	Communication	Establishment & control	COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)

Inter-face	Interface description			
	Functional grouping	Function class	Entities involved	Interfaces involved
		Addressing & routing	COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
		Message transport	COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
		Service announcement & detection	COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
	Context	Provider configuration	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
		Context acquisition & change notification	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
	Interface control	Scanning & locating	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
		Registration	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
		Marshalling	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-

Inter-face	Interface description			
	Functional grouping	**Function class**	**Entities involved**	**Interfaces involved**
				SDA)
		Composition	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
	Algorithmic & procedural control	Capability exchange	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)
		Configuration primitives	CE (CPM, CSC, CCG), FU, COM, TRA	CE-CE (CE-CPM, CPM-CSC, CE-CCG, CE-CSC, CPM-CCG) CE-F (CPM-PP, CSC-SDA)

Table 7-1: Interface categories related to self-growing.

Interfaces CE-CPM, CE-CCG, and CE-CSC are a subset of the general interface CE-CE in the sense that the involved entities are different types of CEs in the sense that they have the same internal structure but have dedicated operational goals. The following table presents a mapping of the functionality provided by these interfaces in relation to the self-growing functions presented in section 3.

Interface	Self-growing Functional Grouping	Self-growing Function Class
CE-CPM	Knowledge provider	Rule base
		Policy provider
	Control & configuration service	Knowledge provider management services
CE-CCG	Communication service provider	Addressing & routing
		Message transport service
		Security services
	Control & configuration service	Cognitive control and management
CE-CSC	Control & configuration service	Context provider/sensor configuration

Table 7-2: Mapping of self-growing functions to CE-CPM, CE-CSC, and CE-CCG interfaces.

Moreover, interfaces CPM-PP and CSC-SDA are a subset of the general CE-F interface since the Policy Provider and the Sensor Data Aggregator are different realizations of an FU. Their exact specification will be provided in a future deliverable of the project.

7.2 Data structures - Self-growing descriptors

The self-growing descriptor introduced herein is a data structure describing the self-growing attribute of a node or network. The definition of this data structure is in an early stage and will be refined by subsequent documents. It will be specified in detail using XML and UML.

The self-growing descriptor can be utilized as a descriptive data structure associated with a dedicated entity (node or network e.g., as an electronic datasheet or as a capability set) or can be communicated in part or in whole between the entities requesting certain behaviour (e.g. as a configuration parameter set) from the original entity. When used in communication the self-growing descriptor may be partially communicated:

- It may be merged with existing information by the receiving entity optimizing, for example the communication overhead by avoiding repeated transmission of the same contents, chaining subsequent communication messages each carrying distinct parts of the descriptor,
- It may augment or replace existing information at the receiving entity and may update existing descriptors.

Additionally, descriptors may be modified by certain commands (e.g. messages), referencing uniquely identifiable elements of the descriptor such as parameters or rules and removing, disabling or enabling these elements. This capability also characterises a self-growing descriptor as a configuration parameter set. Hence, a self-growing descriptor as well as its key elements must be uniquely identifiable (both in a local and global context if needed) and must allow to be bound with a certain device, device class, network, network class, or application, in general setting the applicable scope of uniqueness of a descriptor. If two or more nodes or networks join in a self-growing collaboration, a newly generated (e.g. merged) and unique self-growing descriptor may replace (i.e. temporarily overlay) the descriptors of the joining nodes or networks.

A self-growing descriptor consists of two top-level key elements: The Dictionary and the LifeCycle elements, see Figure 7-1. In addition, the elements MatRules, MatParameters, MatPurposes extends the self-growing descriptor to capture self-learned rules, parameters and self-configured purposes forming the basis for life cycle modifications e.g., in responding to an incident. These elements are mandatory but may contain no subsequent elements.

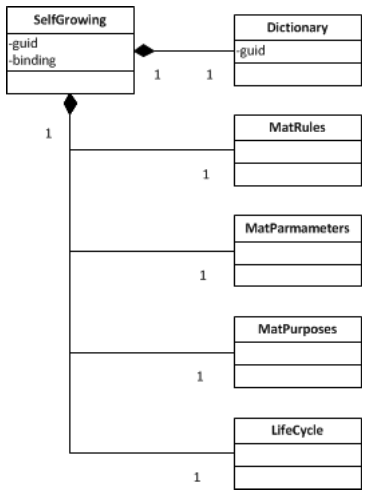

Figure 7-1: Self-growing descriptor: top-level data structure.

The **Dictionary** element in Figure 7-2 serves two main purposes. First, it describes the attributes and capacities of the entity (i.e. the node, collection of nodes or network) in terms of realisable parameters and rules. It also provides a searchable database suitable for comparing nodes or networks for compatibility prior to joining in a self-growing manner. The Dictionary element is designed to enable decision-making, serving as a part of the ontology of the cognitive engines realising decisions. Thus, merging of dictionaries is a common operation in self-growing.

The Dictionary is defining possible purposes and progressions in a parameterized form. That is, it specifies configuration parameters, but does not specify the values of these parameters. This is achieved by providing a LifeCycle element.

Parameters summarized in a Dictionary element are described by name, class, unique identity, metrics, limits and similar information that allow to verify if two parameters of, potentially, different self-growing descriptors match in describing the same attribute using compatible metrics and can, if used as a configuration parameter, attain an overlapping range of possible parameter values. In addition, Rules and Purposes (see below) that rely on this parameter are cross-referenced for convenience, assisting cognitive engines in determining the scope of their decisions.

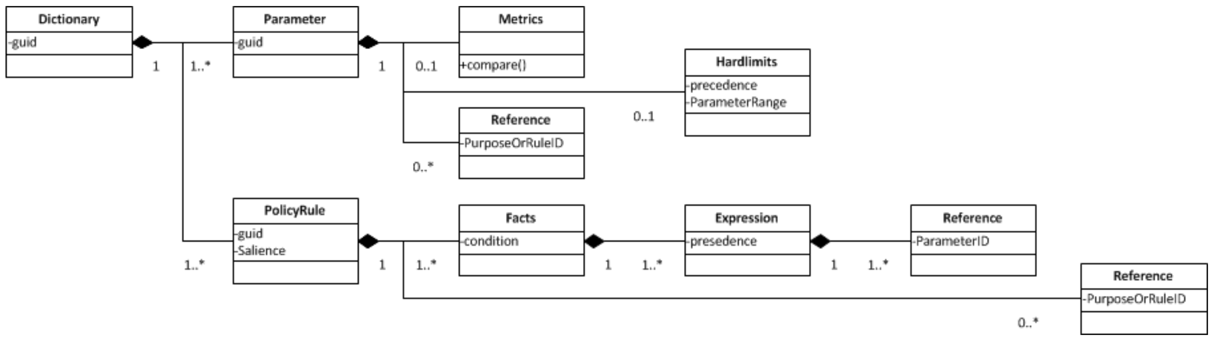

Figure 7-2: Self-growing descriptor: Dictionary.

PolicyRules summarized in a Dictionary element are identified by a unique id and in total provide the rule set that can be utilized to implement a self-growing lifecycle. Assuming an inference engine, a single PolicyRule element is flexible and can implement either a policy or a progression rule depending on the interpretation of its expression and salience by the inference engine.

- If understood as a progression rule, a PolicyRule completely describes attainable purposes of any possible lifecycle. Since purposes are described by a certain set of parameter values, attainable purposes are fixed by all progression rules resulting in the same set of parameters,
- If understood as a policy, a PolicyRule describes the operating range by all parameter value sets attainable. In addition to a progression rule, a policy adds the information if the resulting parameter value set prohibits or allows a certain purpose.

Both Parameter and PolicyRule may contain elements that reference external parameters or rules (i.e. not included in this self-growing descriptor), enabling to develop and use global shared dictionaries to reduce complexity for large-scale systems.

The LifeCycle element in Figure 7-3 links the Parameters and PolicyRules summarized by the Dictionary element and specifies the self-growing attribute of the associated entity by defining attainable purposes and progressions. In addition, it can reference local rules, parameters or purposes provided by the MatRules, MatPurposes and MatParameters elements to include self-learned attributes into a lifecycle, for example.

The LifeCycle element defines **Purposes** by giving associated Parameters and assigning values to these parameters. The resulting set of parameter values defines the configuration required to realise a target purpose for the given entity. It may also reference learned or dynamically created Purposes. The latter allows modifying lifecycles dynamically in consequence of applying an Exception rule, for example.

The LifeCycle element also defines **Progressions** in terms of rules to apply in a certain context as a **Transition** or **Exception**. Transitions are applied under regular conditions to progress from an initial Purpose to a next Purpose. Exceptions may apply under special conditions and cause progression from an initial Purpose to an exceptional Purpose. Optionally, a Purpose that allows re-entering the initial life cycle after an exception has been handled can be given here.

A Purpose or a Transition may be denoted as an exception itself. That is, it can be a progression leading into a terminal purpose when taken, can be a terminal purpose, or can be in conflict with other progressions or purposes. The latter usually disables Purposes or Transitions referenced by the element.

In addition, the LifeCycle element may contain references to Purposes included in the MatPurposes element or to rules included in the MatRules element. The latter may reference parameters included in the MatParameters element only in order to avoid potential inconsistencies when 'static' Transitions or Purposes reference 'dynamic' parameters.

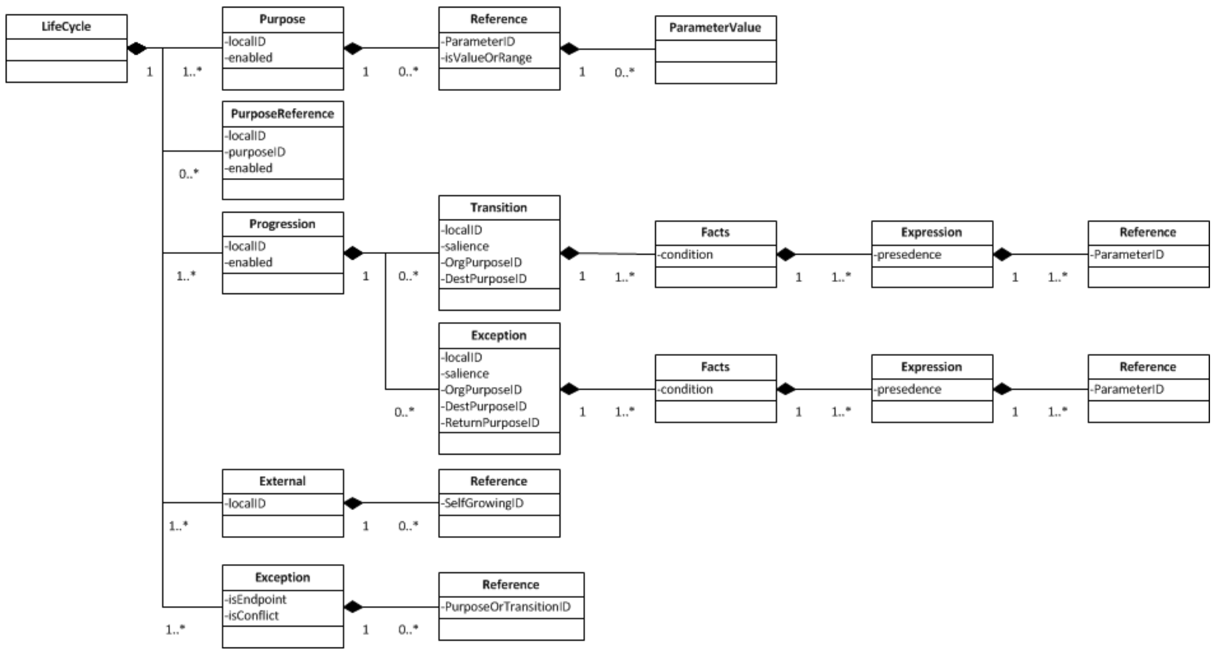

Figure 7-3 Self-growing descriptor: LifeCycle.

The **MatRules** (Figure 7-4), **MatParameters** (Figure 7-5) and **MatPurposes** (Figure 7-6) elements have been included into the self-growing descriptor to enable flexible dynamic creation and modification of life cycles. The terminology is derived from the word 'maturity' in order to emphasize the self-learning character of rules, parameters and purposes covered by these elements. These elements enable creation of temporary purposes and lifecycles due to collaboration of self-growing nodes or networks.

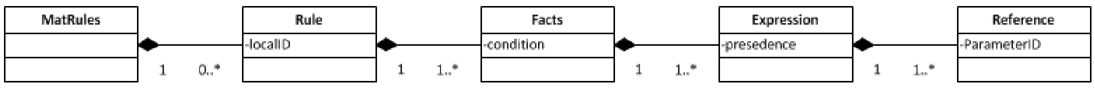

Figure 7-4 Self-growing descriptor: MatRules.

More generally, they provide an initial approach to handle the problem of disruptions of the planned lifecycle due to exceptions (e.g. for the 'emergency use case' UC-09) or for critical resource depletion situations e.g., for the 'WSN collaboration' UC-08. If such a situation is encountered, a cognitive engine may disable (partly or in whole) existing purposes, transitions or exceptions and may compose a new lifecycle by selecting suitable purposes and transitions.

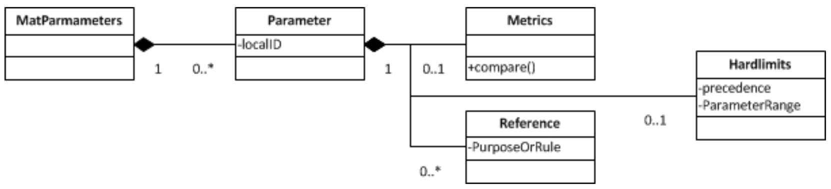

Figure 7-5 Self-growing descriptor: MatParameters.

It should be noted here that the current level of detail of this specification is not sufficient to realise this approach. This is due to missing Dictionary elements describing the cost of a purpose (with respect to some evaluation metric also not yet specified) in order to select the most appropriate purposes from the existing lifecycle or tune existing parameters to create a Purpose attainable within the current context. Since this is closely related to the cognitive engine design, it is still considered work in progress.

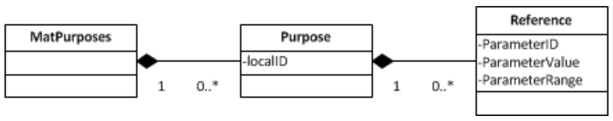

Figure 7-6 Self-growing descriptor: MatPurposes.

8. CONSERN Entity Interactions and Diagrams

8.1 Identification of Interfaces

The CONSERN approach for identifying interfaces is closely aligned to the ETSI RRS approach as used in [8]. As such, it will allow for a straightforward communication and possible exploitation by those standards bodies. Based on the CONSERN functional architecture, which is detailed in chapter 5, suitable points of attachment are identified and related interfaces are introduced between CONSERN Cognitive Engines. The corresponding details are given in chapter 4 of this document, see in particular Figure 4-1 and Figure 4-2. The relevant information model definitions are introduced in section 7.2. By building on these previous results, this section details basic service types as they were introduced in chapter 7. Message Sequence Charts (MSCs) are introduced in order to detail the interactions between concerned entities. The interfaces defined herein should be seen as preliminary proposals based on the knowledge available today and they may be updated and/or refined in later steps of the CONSERN evolution.

In the sequel the services required are introduced:

8.1.1 *Control & Status: Association*

Objective: The *Association* service initiates the exchange of information required for setting up the data exchange between two CONSERN Cognitive Engines.

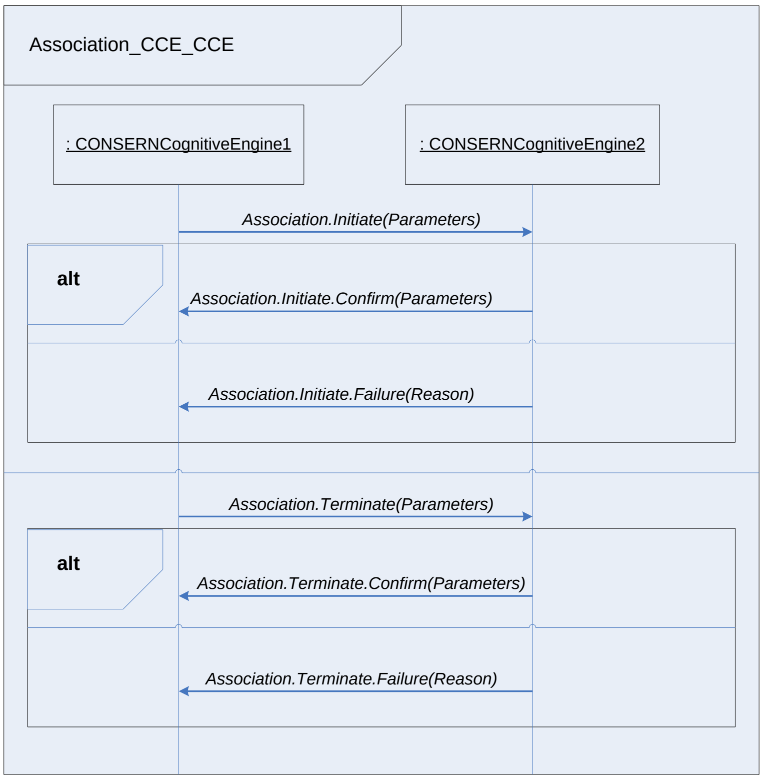
Association_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
Association.Initiate(Parameters)
alt
Association.Initiate.Confirm(Parameters)
Association.Initiate.Failure(Reason)
Association.Terminate(Parameters)
alt
Association.Terminate.Confirm(Parameters)
Association.Terminate.Failure(Reason)

Figure 8-1: Service for set-up of an association between two CONSERN Cognitive Engines.

- The ***Association.Initiate*** service allows a CONSERN Cognitive Engine to trigger the set-up of an association with one or several other CONSERN Cognitive Engines. The parameters delivered with the ***Association.Initiate*** service, preferred configuration parameters, etc. can be requested.

- The **Association.Terminate** service allows a CONSERN Cognitive Engine to terminate an association with one or several other CONSERN Cognitive Engines. The parameters delivered with the **Association.Terminate** service, information related to the termination can be delivered.

8.1.2 *Control & Status: Authentication & trust establishment*

Objective: The *Authentication & trust establishment* service initiates the exchange of information required for setting up a security framework between two CONSERN Cognitive Engines.

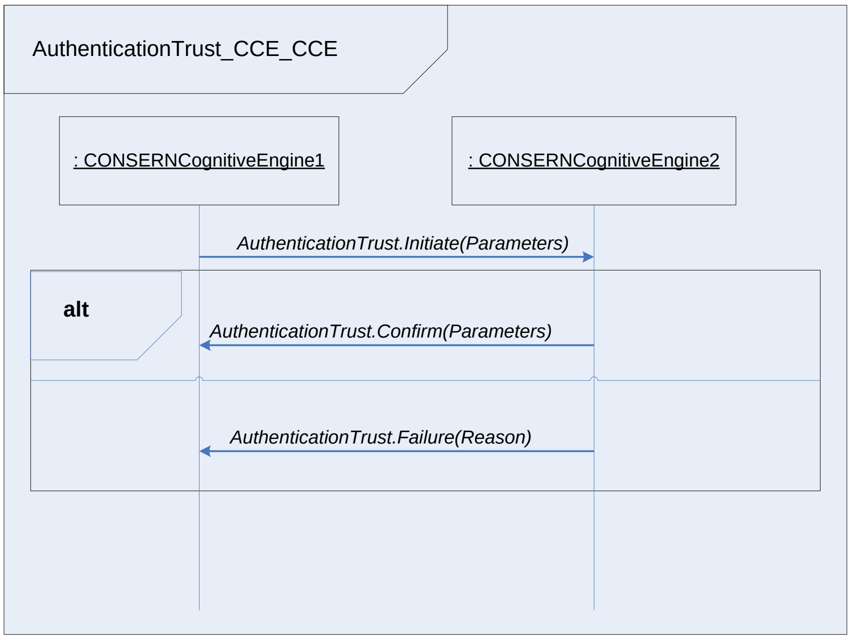
AuthenticationTrust_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
AuthenticationTrust.Initiate(Parameters)
alt
AuthenticationTrust.Confirm(Parameters)
AuthenticationTrust.Failure(Reason)

Figure 8-2: Service for set-up of an Authentication and Trust relationship between two CONSERN Cognitive Engines.

- The ***AuthenticationTrust.Initiate*** service allows a CONSERN Cognitive Engine to trigger the set-up of a trust relationship with one or several other CONSERN Cognitive Engines. The parameters delivered with the ***AuthenticationTrust.Initiate*** service specify the (security) level of the requested trust relation as well as required information to establish the latter.

8.1.3 *Control & Status: Admission Control*

Objective: The *Admission Control* service allows controlling access rights to a CONSERN Cognitive Engine.

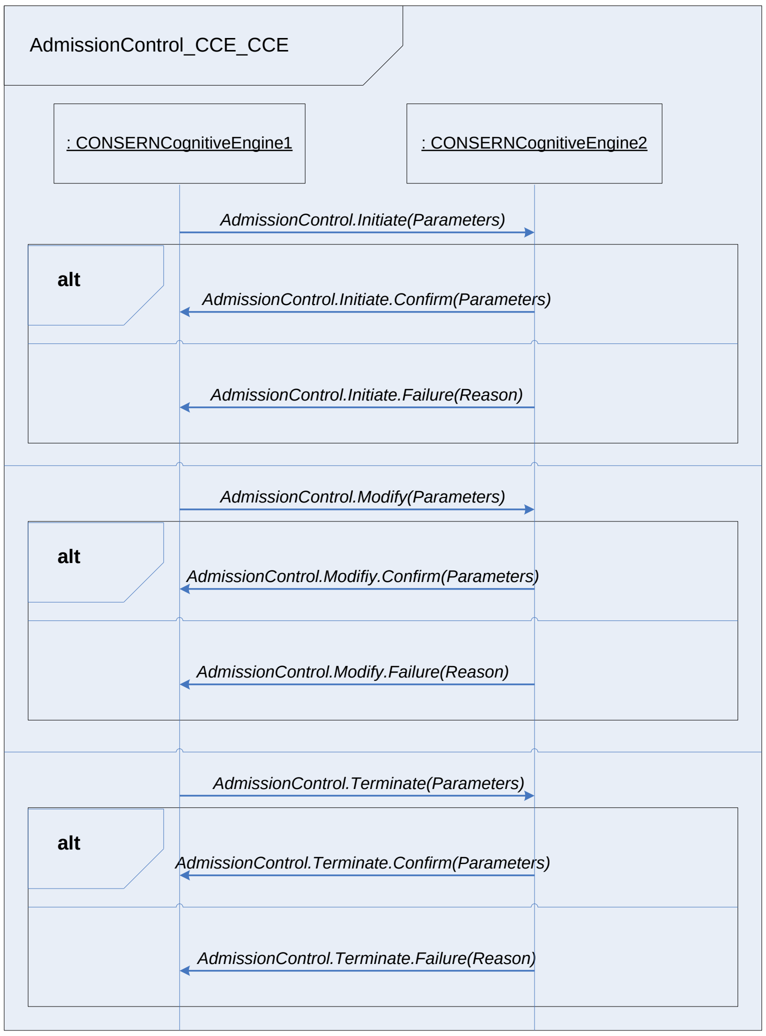
AdmissionControl_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
AdmissionControl.Initiate(Parameters)
alt
AdmissionControl.Initiate.Confirm(Parameters)
AdmissionControl.Initiate.Failure(Reason)
AdmissionControl.Modify(Parameters)
alt
AdmissionControl.Modifiy.Confirm(Parameters)
AdmissionControl.Modify.Failure(Reason)
AdmissionControl.Terminate(Parameters)
alt
AdmissionControl.Terminate.Confirm(Parameters)
AdmissionControl.Terminate.Failure(Reason)

Figure 8-3: Service for set-up of an Admission Control Framework between two CONSERN Cognitive Engines.

- The **AdmissionControl.Initiate** service allows a CONSERN Cognitive Engine to initially request controlling access rights for one or several other CONSERN Cognitive Engines. The parameters delivered with the **AdmissionControl.Initiate** service, specify the requested access rights.
- The **AdmissionControl.Modify** service allows a CONSERN Cognitive Engine to request a modification of access rights previously requested and confirmed by either a preceding **AdmissionControl.Modify** or **AdmissionControl.Initiate** service request.
- The **AdmissionControl.Terminate** service allows a CONSERN Cognitive Engine to request a revocation of previously granted access rights.

8.1.4 *Configuration: Capability Exchange*

Objective: The *Capability Exchange* service allows exchanging capability related information between two CONSERN Cognitive Engines.

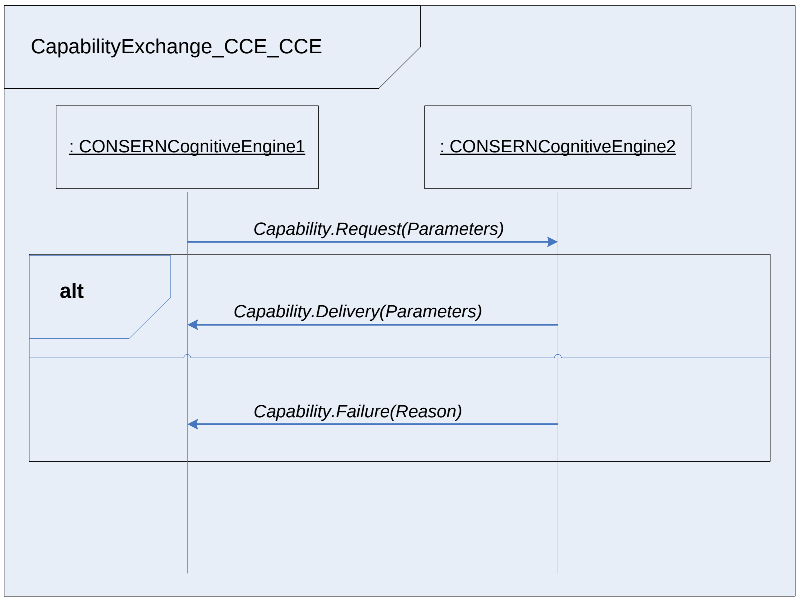
CapabilityExchange_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
Capability.Request(Parameters)
alt
Capability.Delivery(Parameters)
Capability.Failure(Reason)

Figure 8-4: Service for Capability Exchange between two CONSERN Cognitive Engines.

- The **Capability.Request** service allows a CONSERN Cognitive Engine to request the exchange of current settings of capabilities with one or several other CONSERN Cognitive Engines for their capabilities. The parameters delivered with the Capability.Request service, specify the capabilities of interest for this query.

8.1.5 *Configuration: Configuration Primitives*

Objective: The *Configuration Primitives* service allows requesting information related to available configuration primitives from a CONSERN Cognitive Engine.

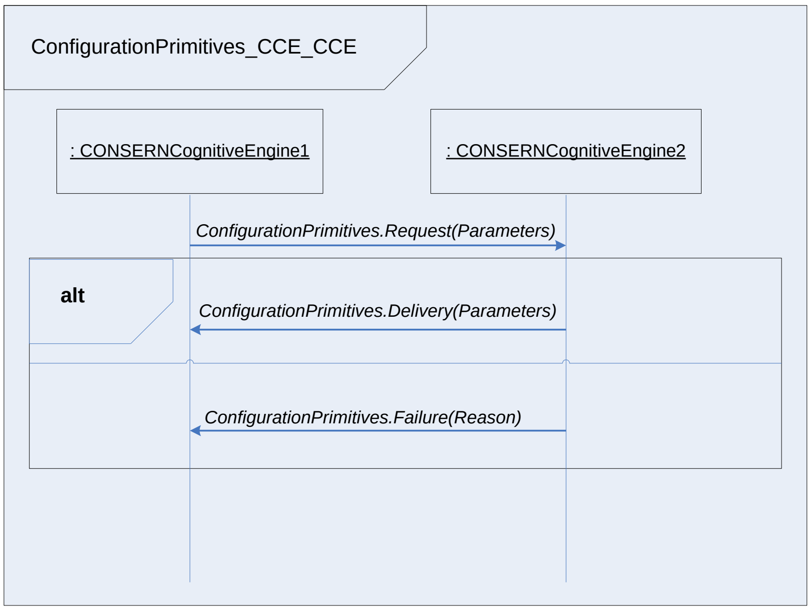
ConfigurationPrimitives_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
ConfigurationPrimitives.Request(Parameters)
alt
ConfigurationPrimitives.Delivery(Parameters)
ConfigurationPrimitives.Failure(Reason)

Figure 8-5: Service for requesting Configuration Primitives from a CONSERN Cognitive Engine.

- The **ConfigurationPrimitives.Request** service allows a CONSERN Cognitive Engine to query one or several other CONSERN Cognitive Engines for their capabilities. The parameters delivered with the **ConfigurationPrimitives.Request** service, specify the capabilities of interest for this query.

8.1.6 *Communication: Establishment & Control*

Objective: The *Establishment and Control* service allows to set-up a data link and to perform the corresponding control actions between two CONSERN Cognitive Engines.

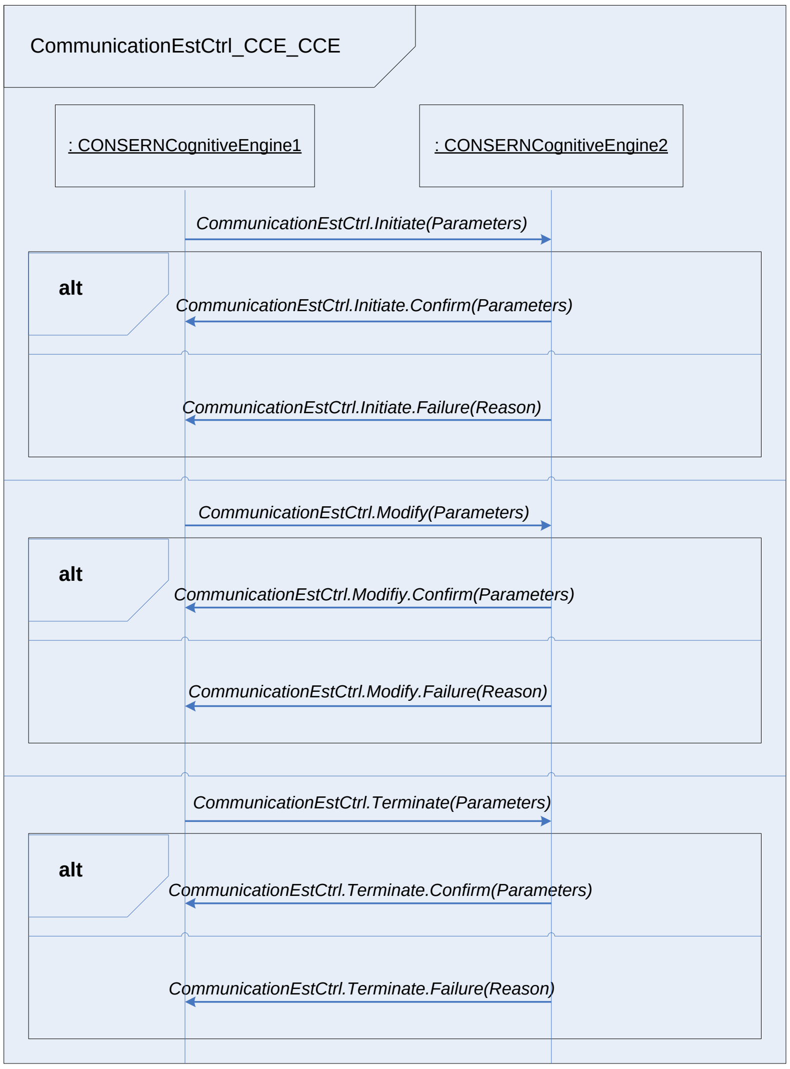
CommunicationEstCtrl_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
CommunicationEstCtrl.Initiate(Parameters)
alt
CommunicationEstCtrl.Initiate.Confirm(Parameters)
CommunicationEstCtrl.Initiate.Failure(Reason)
CommunicationEstCtrl.Modify(Parameters)
alt
CommunicationEstCtrl.Modifiy.Confirm(Parameters)
CommunicationEstCtrl.Modify.Failure(Reason)
CommunicationEstCtrl.Terminate(Parameters)
alt
CommunicationEstCtrl.Terminate.Confirm(Parameters)
CommunicationEstCtrl.Terminate.Failure(Reason)

Figure 8-6: Service for Establishment and Control of a data link between two CONSERN Cognitive Engines.

- The **CommunicationEstCtrl.Initiate** service allows a CONSERN Cognitive Engine to request the initialization of a communication and control path between CONSERN Cognitive Engines. The parameters delivered with the **CommunicationEstCtrl.Initiate** service specify the involved CONSERN Cognitive Engines and possibly parameters specifying the kind of message delivery service established or specific routing preferences.
- The **CommunicationEstCtrl.Modify** service allows a CONSERN Cognitive Engine to request a modification the communication and control previously requested and confirmed by either a preceding **CommunicationEstCtrl.Modify** or **CommunicationEstCtrl.Initiate** service request.
- The **AdmissionControl.Terminate** service allows a CONSERN Cognitive Engine to request a revocation of previously established communication and control paths between CONSERN Cognitive Engines.

8.1.7 *Communication: Addressing & Routing*

Objective: The *Addressing and Routing* service allows to set-up and modify addressing and routing parameters for the communication between two CONSERN Cognitive Engines.

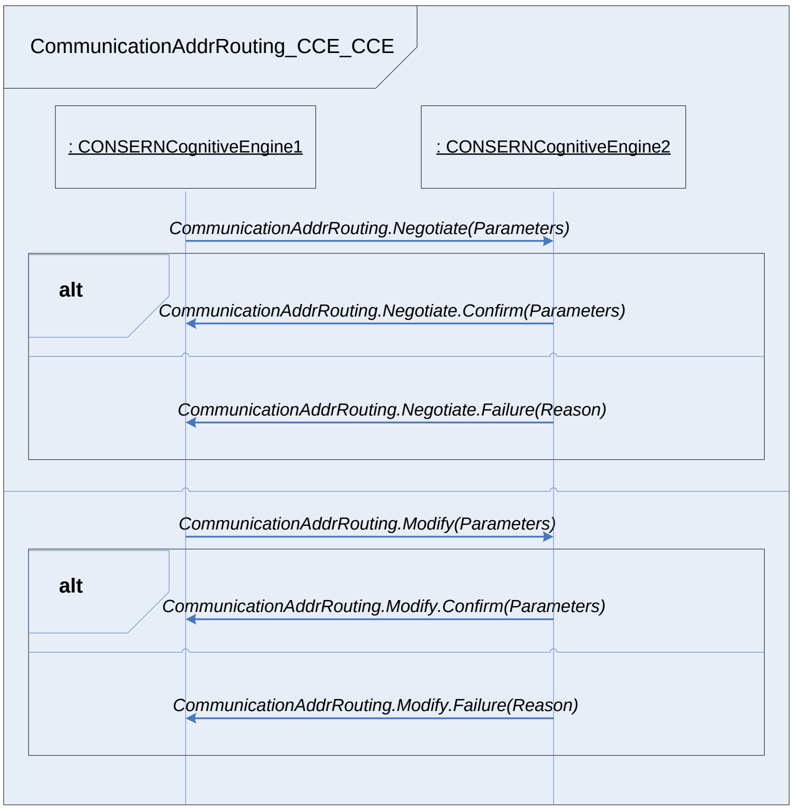
CommunicationAddrRouting_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
CommunicationAddrRouting.Negotiate(Parameters)
alt
CommunicationAddrRouting.Negotiate.Confirm(Parameters)
CommunicationAddrRouting.Negotiate.Failure(Reason)
CommunicationAddrRouting.Modify(Parameters)
alt
CommunicationAddrRouting.Modify.Confirm(Parameters)
CommunicationAddrRouting.Modify.Failure(Reason)

Figure 8-7: Service for negotiating addressing and routing for the communication between two CONSERN Cognitive Engines.

- The **CommunicationAddrRouting.Negotiate** service allows a CONSERN Cognitive Engine to negotiate with one or several other CONSERN Cognitive Engines addressing schemes and routing preferences for communication between CONSERN entities. Those preference requests are forwarded by the CONSERN Cognitive Engine to communication modules local to each CONSERN entity.
- The **CommunicationAddrRouting.Modify** service allows a CONSERN Cognitive Engine to modify previously negotiated addressing schemes and routing preferences via a preceding invocation of the **CommunicationAddrRouting.Modify** or **CommunicationAddrRouting.Negotiate** service.

8.1.8 *Communication: Message Transport*

Objective: The *Message Transport* service allows exchanging a message between two CONSERN Cognitive Engines.[6]

[6] Note that the realization of this service has to be provided by the communication module when it comes to an implementation. This document defines the available services / functionality among CCEs.

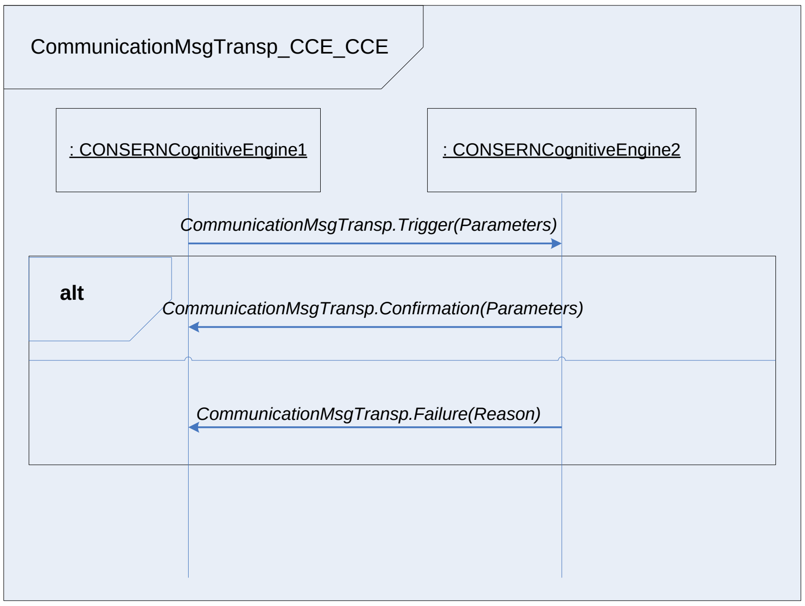
CommunicationMsgTransp_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
CommunicationMsgTransp.Trigger(Parameters)
alt
CommunicationMsgTransp.Confirmation(Parameters)
CommunicationMsgTransp.Failure(Reason)

Figure 8-8: Service for Message Transport between two CONSERN Cognitive Engines.

- The **CommunicationMsgTransp.Trigger** service allows a CONSERN Cognitive Engine to send a message to another CONSERN Cognitive Engine. The parameters delivered with the **CommunicationEstCtrl.Initiate** service specify at least the involved CONSERN Cognitive Engines the message is to be delivered to.

8.1.9 *Communication: Service Announcement & Detection*

Objective: The *Service Announcement & Detection* service allows to make the availability of services by a CONSERN Cognitive Engines known, either using push or pull mechanisms.

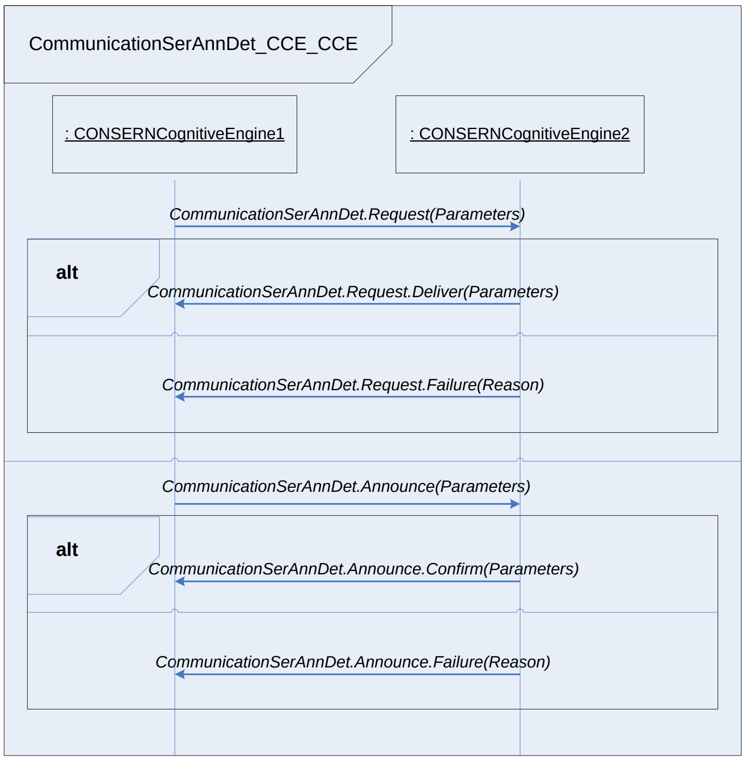
CommunicationSerAnnDet_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
CommunicationSerAnnDet.Request(Parameters)
alt
CommunicationSerAnnDet.Request.Deliver(Parameters)
CommunicationSerAnnDet.Request.Failure(Reason)
CommunicationSerAnnDet.Announce(Parameters)
alt
CommunicationSerAnnDet.Announce.Confirm(Parameters)
CommunicationSerAnnDet.Announce.Failure(Reason)

Figure 8-9: Service for Service Announcement & Detection as made available by a CONSERN Cognitive Engine.

- The **CommunicationSerAnnDet.Request** service allows a CONSERN Cognitive Engine to request one or several other CONSERN Cognitive Engine to announce their availability in providing services. The parameters delivered with the **CommunicationSerAnnDet.Request** service specify at least the involved CONSERN Cognitive Engines.
- The **CommunicationSerAnnDet.Announce** service allows a CONSERN Cognitive Engine to announce services being provided. The parameters delivered with the **CommunicationSerAnnDet.Announce** service specify at least available services.

8.1.10 *Context: Provider Configuration*

Objective: The *Provider Configuration* service allows obtaining the provider's configuration parameters for a CONSERN Cognitive Engine.

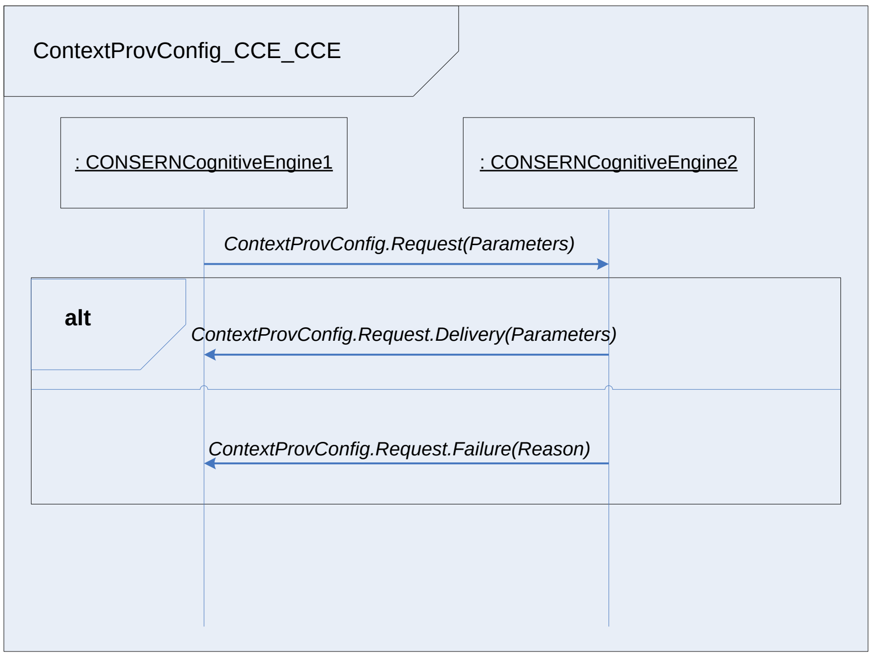
ContextProvConfig_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
ContextProvConfig.Request(Parameters)
alt
ContextProvConfig.Request.Delivery(Parameters)
ContextProvConfig.Request.Failure(Reason)

Figure 8-10: Service for Provision of Provider's Context Information as made available by a CONSERN Cognitive Engine.

- The **ContextProvConfig.Request** service allows a CONSERN Cognitive Engine to request one or several other CONSERN Cognitive Engine to report information on their provider specific configuration. Parameters include a description of the parameters of interest.

8.1.11 *Context: Context acquisition & change notification*

Objective: The *Context acquisition & change notification* service allows to update context information delivered by a CONSERN Cognitive Engine either by push or pull mechanisms.

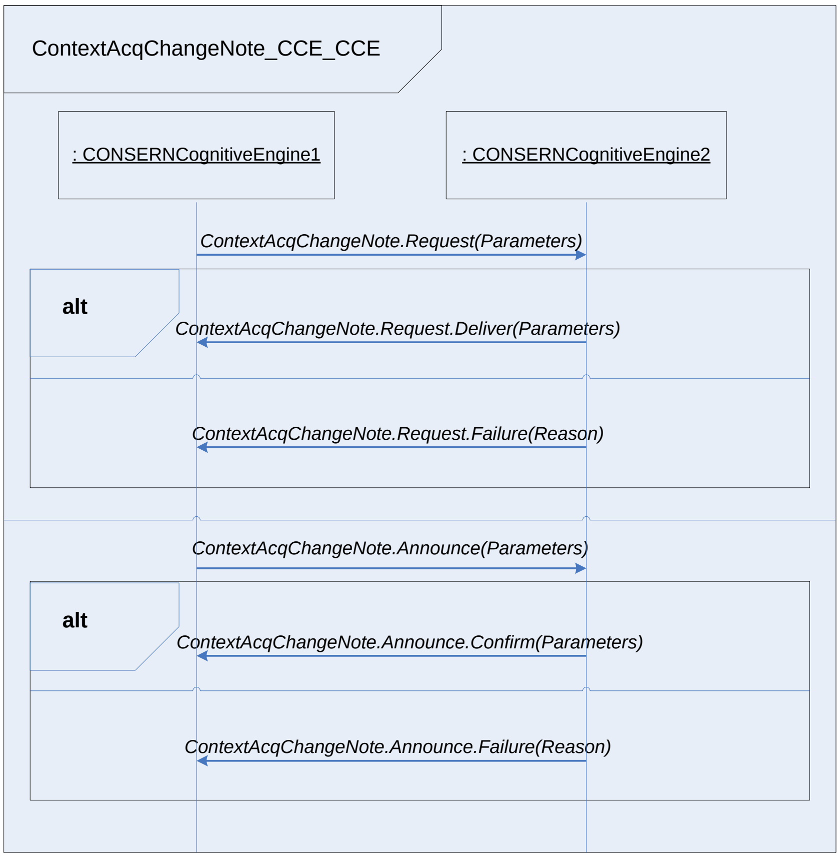
ContextAcqChangeNote_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
ContextAcqChangeNote.Request(Parameters)
alt
ContextAcqChangeNote.Request.Deliver(Parameters)
ContextAcqChangeNote.Request.Failure(Reason)
ContextAcqChangeNote.Announce(Parameters)
alt
ContextAcqChangeNote.Announce.Confirm(Parameters)
ContextAcqChangeNote.Announce.Failure(Reason)

Figure 8-11: Service for provision of Context Acquisition & Change Notification as made available by a CONSERN Cognitive Engine.

- The **ContextAcqChangeNote.Request** service allows a CONSERN Cognitive Engine to request one or several other CONSERN Cognitive Engine (periodically) announce context information. The parameters delivered with the **ContextAcqChangeNote.Request** service specify criteria for CONSERN Cognitive Engines requested to report context as well as a description of the kind of requested context information.
- The **ContextAcqChangeNote.Announce** service allows a CONSERN Cognitive Engine to announce context information previously requested by **ContextAcqChangeNote.Request**. The parameters delivered with the **ContextAcqChangeNote.Announce** contain information on the announced context.

8.1.12 *Context: Consumer Configuration*

Objective: The *Consumer Configuration* service allows obtaining the consumer's configuration parameters for a CONSERN Cognitive Engine.

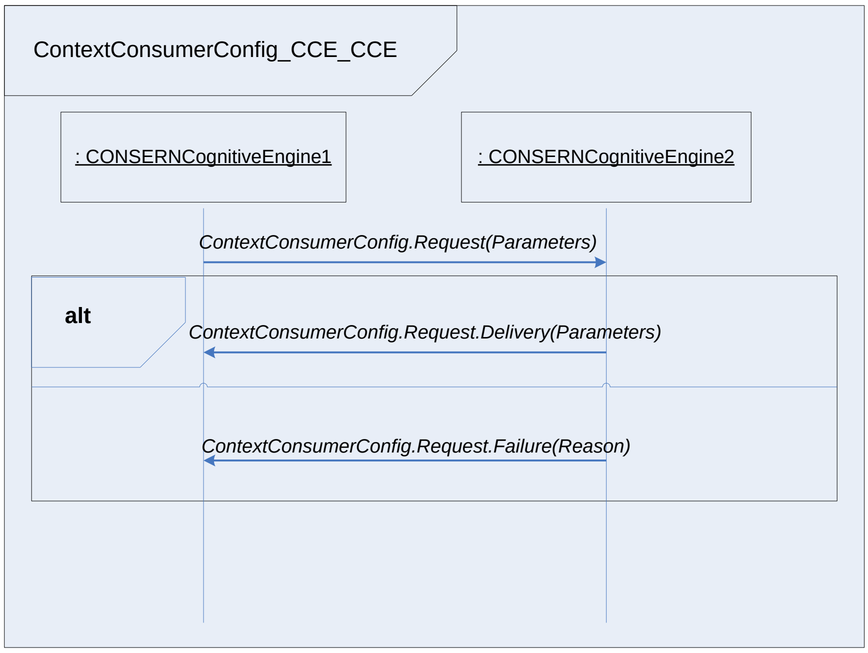
ContextConsumerConfig_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
ContextConsumerConfig.Request(Parameters)
alt
ContextConsumerConfig.Request.Delivery(Parameters)
ContextConsumerConfig.Request.Failure(Reason)

Figure 8-12: Service for provision of Consumer Configuration as made available by a CONSERN Cognitive Engine.

- The **ContextConsumerConfig.Request** service allows a CONSERN Cognitive Engine to request one or several other CONSERN Cognitive Engine to report information on their consumer specific configuration. Parameters include a description of the parameters of interest.

8.1.13 *Cognition Control: Knowledge Exchange*

Objective: The *Knowledge Exchange* service allows obtaining the exchange knowledge related information between two CONSERN Cognitive Engines.

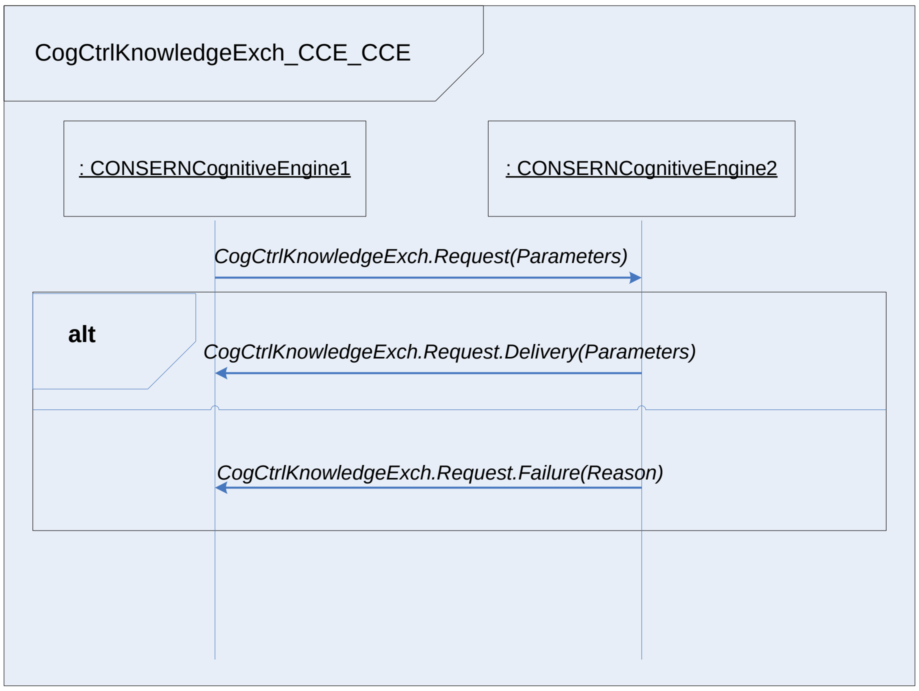
CogCtrlKnowledgeExch_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
CogCtrlKnowledgeExch.Request(Parameters)
alt
CogCtrlKnowledgeExch.Request.Delivery(Parameters)
CogCtrlKnowledgeExch.Request.Failure(Reason)

Figure 8-13: Service for Knowledge Exchange as made available by a CONSERN Cognitive Engine.

- The **CogCtrlKnowledgeExch.Request** service allows a CONSERN Cognitive Engine to request one or several other CONSERN Cognitive Engine to report information on their obtained knowledge for fulfilling a specific service. Parameters include a description of the parameters of interest as well as a description of the CONSERN Cognitive Engines of interest.

8.1.14 *Cognition Control: Negotiation*

Objective: The *Negotiation* service allows to perform negotiation actions between two CONSERN Cognitive Engines, for example related to the definition of common configuration parameters, etc.

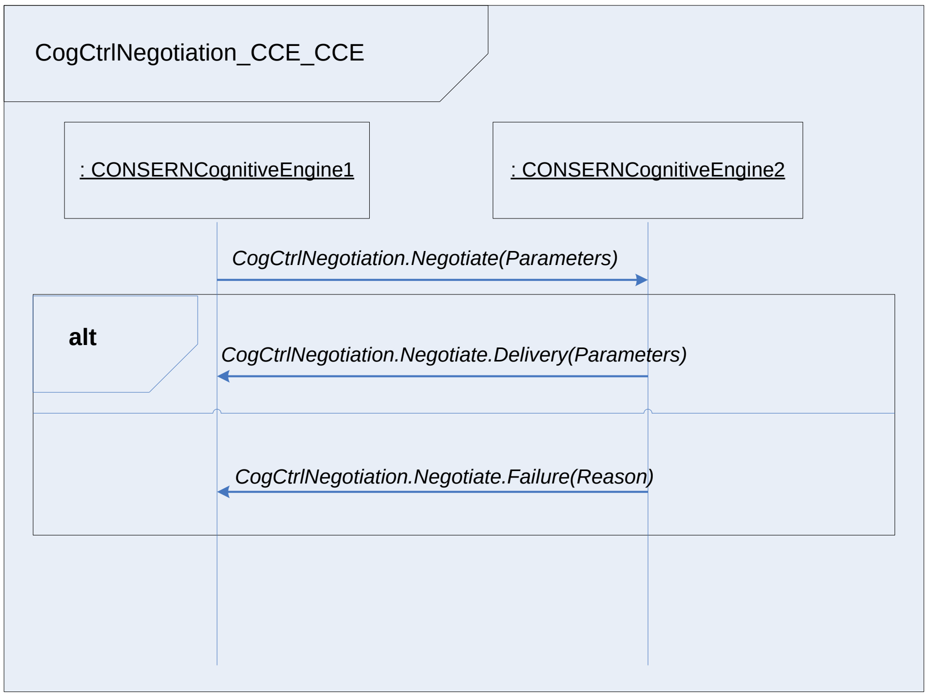
CogCtrlNegotiation_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
CogCtrlNegotiation.Negotiate(Parameters)
alt
CogCtrlNegotiation.Negotiate.Delivery(Parameters)
CogCtrlNegotiation.Negotiate.Failure(Reason)

Figure 8-14: Service for Negotiation as made available by a CONSERN Cognitive Engine.

- The **CogCtrlNegotiation.Negotiate** service allows a CONSERN Cognitive Engine to negotiate actions or setting of common or dependent configuration parameters with one or several other CONSERN Cognitive Engines. Parameters include a description of the configuration / action under negotiation as well as a description of the CONSERN Cognitive Engines of interest.

8.1.15 *Cognition Control: Policy exchange & enforcement*

Objective: The *Policy exchange & enforcement* service allows to introduce and to enforce policies to be used by a given CONSERN Cognitive Engine.

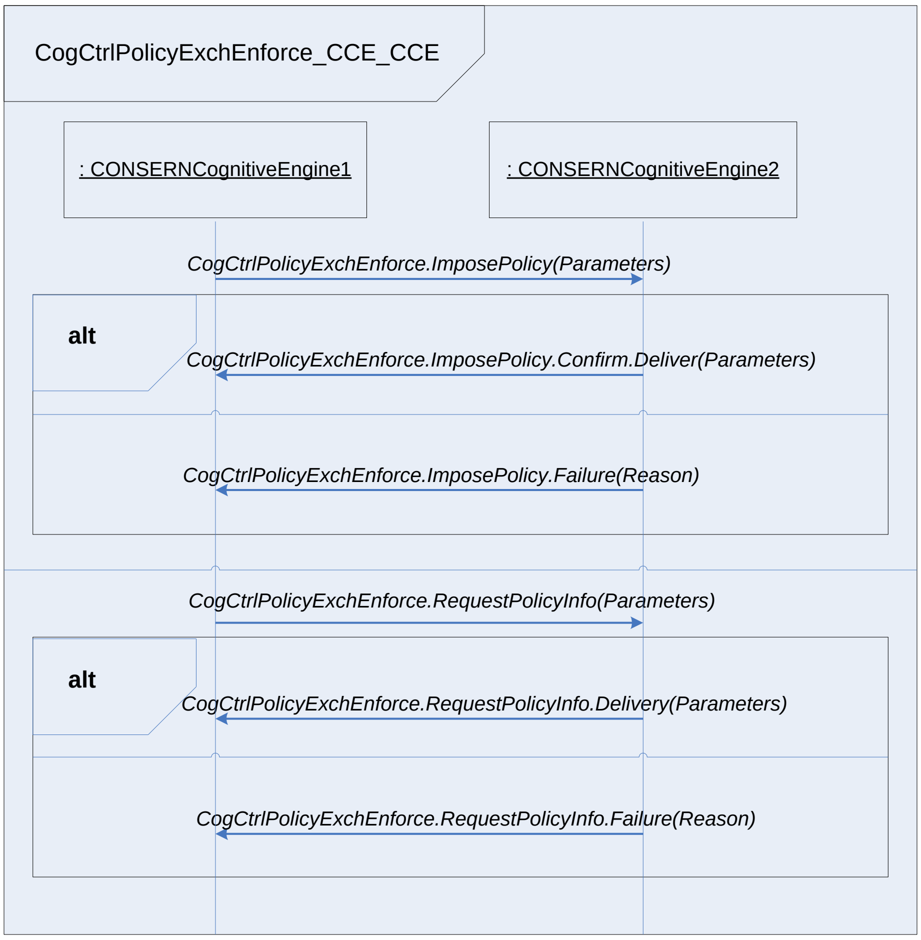
CogCtrlPolicyExchEnforce_CCE_CCE
: CONSERNCognitiveEngine1
: CONSERNCognitiveEngine2
CogCtrlPolicyExchEnforce.ImposePolicy(Parameters)
alt
CogCtrlPolicyExchEnforce.ImposePolicy.Confirm.Deliver(Parameters)
CogCtrlPolicyExchEnforce.ImposePolicy.Failure(Reason)
CogCtrlPolicyExchEnforce.RequestPolicyInfo(Parameters)
alt
CogCtrlPolicyExchEnforce.RequestPolicyInfo.Delivery(Parameters)
CogCtrlPolicyExchEnforce.RequestPolicyInfo.Failure(Reason)

Figure 8-15: Service for Policy exchange & enforcement by a CONSERN Cognitive Engine.

- The **CogCtrlPolicyExchEnforceImposePolicy** service allows a CONSERN Cognitive Engine to set and enforce a policy / rule set at one or several other CONSERN Cognitive Engines. Parameters include a description of the policy as well as a description of the CONSERN Cognitive Engines of interest.
- The **CogCtrlPolicyExchEnforceRequestPolicyInfo** service allows a CONSERN Cognitive Engine to request information on mandatory (enforced) policy sets from one or several other CONSERN Cognitive Engines. Parameters include a description of the policies of interest as well as a description of the CONSERN Cognitive Engines of interest.

9. Summary and Future Work

This delivery has focussed on providing a basic self-growing architecture. The architecture reflects the requirements earlier derived in D1.1 on a quite high conceptual level and will be further refined in future CONSERN research. The aim is to provide an architecture allowing a practical implementation of a CONSERN system.

Although the main focus has been on highlighting the self-growing aspects the proposed architecture also aims at supporting (possibly with minor modifications) the research made in other WPs i.e., energy efficiency (WP2) and cooperation (WP3). So far no work in this direction has been made mainly because of the numerous numbers of proposals discussed during the development of the architecture.

In the continuation of the project and more specifically in D4.3 the focus will be put on extending the high-level ideas related to self-growing made herein. Focus will here be put on the implementation aspects and in this direction, mechanisms for discovering co-located networks and for negotiating network parameters among such networks in order to enable cross-network sharing of resources will be realized. Moreover, cognitive decision-making strategies will be proposed and a corresponding Rule Base to support the process of joining and splitting networks, as well as to control the target optimisation goal of one or more sub-networks in this process will be developed.

A common problem in typical large scale distributed systems is that as scale increases the total number of interactions among actors that are required to coordinate the behaviour of the distributed system can increase at a much faster rate, depending also on the actual communication model. Therefore, scalability issues may emerge during the evolution of the network in the self-growing process. Thus it is intended to couple self-organization related algorithms, with autonomic capabilities to address this problem. Such algorithms are known to build upon appropriate communication models (asynchronous and ad-hoc) that are inherently efficient, scalar and fault tolerant.

10. References

[1] INFSO-ICT-257542 CONSERN Project, http://www.ict-consern.eu.

[2] CONSERN Deliverable D1.1, “Scenarios, Use Cases, and System Requirements”.

[3] CONSERN Deliverable D4.1, “Initial Description of Self-growing Scenarios, Properties, Requirements and Envisaged Framework”.

[4] 3GPP TS 23.101 V9.0.0 (2009-12), “3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; General Universal Mobile Telecommunications System (UMTS) architecture (Release 9)”.

[5] IEEE Std 754-2008, “IEEE Standard for Floating-Point Arithmetic”, pp. 1—58, 2008.

[6] J. Strassner, E. Lehtihet, N. Agoulmine, “FOCALE - A Novel Autonomic Networking Architecture”.

[7] RDF Working Group, “Description Framework (RDF)”, http://www.w3.org/RDF.

[8] ETSI TR 102 839, “Reconfigurable Radio Systems (RRS); Multiradio Interface for Software Defined Radio (SDR), Mobile Device Architecture and Services”.

[9] Y. Cheng et al., “A generic architecture for autonomic service and network management”, Computer Communications, 2006.