ETSI GR ZSM 009-3 V1.1.1 (2023-08)

Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 3: Advanced topics

Disclaimer

The present document has been produced and approved by the Zero-touch network and Service Management (ZSM) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGR/ZSM-009-3_Cla_AdvTop

Keywords
automation, closed control loop, management, service

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Zero-touch network and Service Management (ZSM).

The present document is part 3 of a multi-part deliverable. Full details of the entire series can be found in part 1 [i.2].

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document investigates advanced topics related to closed-loop operations such as learning and cognitive capabilities (e.g. based on different degrees of use and integration of artificial intelligence technologies), ways to set and evaluate levels of oversight, autonomy, and operational confidence on the behaviour of the closed loops. The present document will document problem statements and technical challenges, derive potential requirements, capture, and evaluate potential solution options, and provide recommendations for further standardization activities.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are not necessary for the application of the present document, but they assist the user with regard to a particular subject area.

[i.1] S. Ayoubi, N. Limam, M.A. Salahuddin, N. Shahriar, R. Boutaba: "Machine Learning for Cognitive Network Management". IEEE Communications Magazine. Vol. 56(1), pp. 158-165, January 2018.

[i.2] ETSI GS ZSM 009-1: "Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 1: Enablers".

[i.3] ETSI GS ZSM 009-2: "Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 2: Solutions for automation of E2E service and network management use cases".

[i.4] ETSI GR ZSM 011 (V1.1.1): "Zero-touch network and Service Management (ZSM); Intent-driven autonomous networks; Generic aspects".

[i.5] TM Forum IG1253 (V1.2.0): "Intent in Autonomous Networks".

[i.6] Russell, Stuart J.; Norvig, Peter (2003) (2nd ed.): "Artificial Intelligence: A Modern Approach". Upper Saddle River, New Jersey: Prentice Hall. Chapter 2. ISBN 0-13-790395-2.

[i.7] ETSI GS ZSM 001: "Zero-touch network and Service Management (ZSM); Requirements based on documented scenarios".

[i.8] Stephen S. Mwanje, Christian Mannweiler: "Towards Cognitive Autonomous Networks" Publisher: WILEY.

[i.9] ETSI GS ZSM 002 (V1.1.1): "Zero-touch network and Service Management (ZSM); Reference Architecture".

[i.10] ETSI TS 128 312 (V17.1.1) (2022-10): "LTE; 5G; Management and orchestration; Intent driven management services for mobile networks (3GPP TS 28.312 version 17.1.1 Release 17)".

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

5G	Fifth Generation of mobile networks
AI	Artificial Intelligence
C-MAPE	Cognitive Control Loop
CL	Closed Loop
CLC	Closed Loop Controller
CPU	Central Processing Unit
IME	Intent Management Entity
IoT	Internet of Things
KPI	Key Performance Indicator
LTE	Long Term Evolution
M2O-CL	Made-to-Order Closed Loop
MD	Management Domain
ML	Machine Learning
OWL	Web Ontology Language
PM	Performance Management
QoE	Quality of Experience
RDF	Resource Description Framework
SLA	Service Level Agreement
SON	Self Organizing Networks
TOSCA	Topology and Orchestration Specification for Cloud Applications
TS	Traffic Steering
UE	User Equipment
VNF	Virtual Network Function

4 Next-generation closed-loop operations

It is expected that the future network management systems should be able to handle the increased complexity in the evolving network scenarios arising from diverse network deployments options and massive scalability requirements. As part of the evolution, the management of the network should be automated as much as possible. Accordingly, the management system, specifically the next generation of closed-loop operations, should be able to handle the additional requirements through machine-learning-driven or cognitive automation. Cognition built from learning based on different network and environment events will be used to derive insights and decisions on network and service management scenarios involving single or multiple network events. As described in [i.1] closed loops may benefit from applying machine learning in any of the stages, to make the closed loops more adaptive and self-learning.

As described in [i.8], the evolution towards autonomous networks empowered with cognitive capabilities will happen in stages starting with today's automated network employing SON as closed loops involving a set of automation capabilities configured and managed by the operator. The intermediate stage is the automated network, which adds cognitive capabilities to the management layer thereby eliminating the need for the operator to handle the configuration and management of the network management automation capabilities that constitute the closed loops. The configuration and management for the automation capabilities may be selected based on prevailing contexts in the network environment. Eventually, the operations will evolve further into a fully autonomous network in which cognitive capabilities are introduced into the closed loops of managed and management layers. These learning driven closed loop are then capable of learning the right configuration parameters for different network contexts and update the learning in the context model in a continuous manner. Autonomous networks enable the service user and the network operator as management service consumers to provide their expectations as "intents". The intent-based system converts the intent to objectives which are further processed by different closed loops to satisfy the expectations expressed by the intent. The system accepts input data from different sources to gather different perspective on the network and also the impacts from configuration and environmental changes. This will enable the system to contextualize the information with multi-dimensional perspective of the network behaviour (network performance and user experience) based on different scenarios (configuration, environment, user specific handling, etc.).

This study investigates various challenges in network automation and potential advanced closed loop solutions such as cognitive closed loops, intent based closed loops, dynamic closed loop composition and coordination, inter-dependent closed loop, etc. to support cognitive autonomous network.

5 Problem Statements

5.1 Cognitive closed loops

Closed loops have multiple stages which could benefit from applying Machine Learning (ML) technology. According to the C-MAPE principle [i.1] a cognitive control loop may incorporate ML into any of its stages: ML-based monitoring, analysis, decision making and optimized actions. Such approach makes a CL more adaptive and self-driven/self-learning, as opposed to traditional CLs that are usually self-regulating (implement a pre-defined control logic like in LTE/5G SON) but not self-learning (i.e. they do not change the control logic regardless of the outcome of their actions).

Applying ML within a CL requires the integration of one or more trained ML models in the CL implementation. With state-of-the-art ML methods, an ML model architecture (e.g. neural network layers, activation functions) defines the syntax and semantics of both the input data and the model output. That is, each ML model takes a set of well-defined input data and produces inference results of a specific kind. Due to such focus of a single model, a CL may benefit from having multiple models, each operating on different sets or types of input data and producing different outputs. The reasons for multiple models may include:

- Using multiple models in combination to implement a more complex analytics and decision pipeline (e.g. by considering insights generated by different analytics algorithms in parallel and/or in series). In this case, the involved models are simultaneously active and should be supplied with their respective input data.
- Some models are alternatives to others with different level of functionality or capability. For instance, different models may be created to analyse time series data coming in at different measurement intervals (5 minutes, 15 minutes, 1 hour). Depending on the resolution of the input data available in the deployment of the CL, the corresponding model is selected. Additionally, if the availability of the input data changes (e.g. a data source stops sending data for a period of time), the CL may autonomously switch between models.
- Some models implement special analysis cases that are only needed under specific circumstances. For example, in a hierarchical analysis, high level analysis and decision may be driven by a first model, whereas special non-anticipated cases, such as suspicion of an anomaly, triggers further analysis using other models. As the various models require different input data, the CL may need to activate specific data sources so that they start to produce data required by a newly activated model.

The above considerations forecast a more dynamic interplay between stages of a CL compared to a mostly linear collection-analysis-decision-actuation cycle. Such dynamism also contributes to improved level of autonomy for the CL, such as:

- Proactively re-program data sources to autonomously collect the right type, quality, and quantity of data. The data quality and quantity of data may be a pre-requisite for versatile and high-quality analytics (or to be able to perform analytics at all).
- With enhanced analytics, CLs may implement self-learning capability by analysing the outcome of the actions and learning from a feedback based on how well the actions have reached the goal for which the actions have been initiated.

5.2 Composition and coordination of interdependent closed loops

Multiple closed loops may belong to the same use case with a common goal or related goals. These interdependent CLs may act directly or indirectly on the same managed entity. Hence, the predicted actions of these interdependent CLs may conflict with each other resulting in an undesired operation on the managed entity.

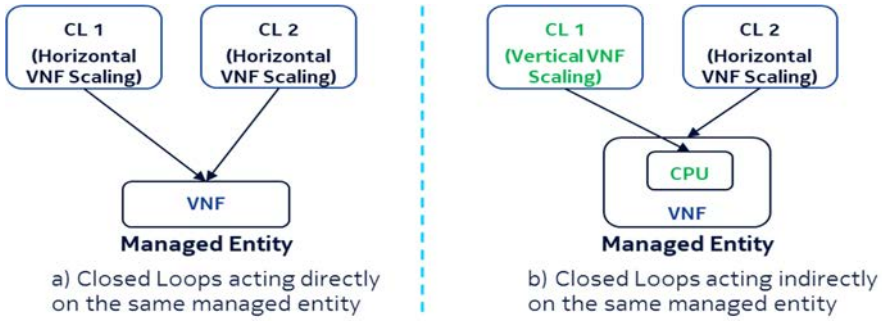

Figure 5.2-1: Example for Interdepended Closed loops

- As shown in figure 5.2-1a), two or more instantiated CLs may act *directly* on the same managed entity, e.g. two CLs acting on the same Virtual Network Function (VNF) instance for horizontal VNF autoscaling.
- As shown in figure 5.2-1b), two or more instantiated CLs may act *indirectly* on the same managed entity, e.g. when one CL acts on the CPU resource of the VNF instance for vertical VNF autoscaling and another CL acts on that same VNF instance for horizontal VNF autoscaling.

Challenges:

Currently, external coordination mechanism (i.e. via CL Coordinator as shown in figure 5.2-1) is employed to detect and mitigate CL conflicts either before (i.e. Pre-action coordination) or immediately after they occur (i.e. Post-action coordination).

In Pre-action coordination, the CL coordinator detects and resolves conflicts between CLs before triggering the Execution stage of the CLs.

In Post-action coordination, the CL coordinator identifies CL actions after the Execution stage that lead to undesired outcomes and determines how to avoid such conflicts in the future.

However, external coordination mechanism assumes that all CLs are independent and can thus be coordinated as independent entities. Consequently, the CLs are currently always composed independently of each other, even though the CLs are interdependent belonging to the same use case with common goal or related goals.

With hundreds or thousands of CLs instantiated in the network, some of these CLs may be nested or hierarchical. Scaling the functionality of the CL Coordinator becomes a challenge.

The potential solutions to address these challenges are specified in clause 6.3. **Related Requirement** are:

- ETSI GS ZSM 001 [i.7] clause 5.2 Req #70 "*ZSM framework shall support the capability of nested closed loops*".

- ETSI GS ZSM 009-2 [i.3] clause 5.2 [CL-general-2] "*The ZSM framework shall allow establishing different types of relationships between Closed Loops. NOTE 2: Examples of relationships are peer Closed Loops, hierarchical or nested Closed Loops.*"

The potential solutions to address these challenges are specified in clause 6.2.

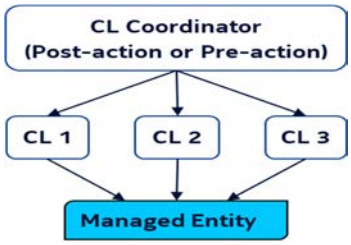

Figure 5.2-2: External Closed Loop Coordinator-1

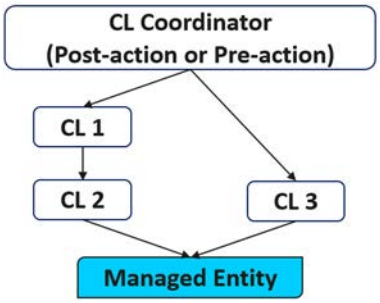

Figure 5.2-3: External Closed Loop Coordinator-2

5.3 Dynamic composition of closed loops

Closed loops may be, in general, within a single management function that combines all the closed loop stages or functionalities. In such cases, multi-vendor interactions need interfaces to the network resources while control and exposure interfaces focus on the capabilities of the combined function.

Alternatively, for multi-vendor network automation environments, closed loops support specific modules for accomplishing specific tasks, e.g. to provide a specific analysis on some data. CLs may be composed from multi-vendors components using standardized services. As illustrated by figure 5.3-1, the different stages can support control and exposure services (marked with "C" and "E" in figure 5.3-1), through which an authorized entity (e.g. the ZSM framework consumer) may control the stages. Each stage also supports the run-time services (marked with D in figure 5.3-1) through which the stages exchange data to accomplish the network management tasks.

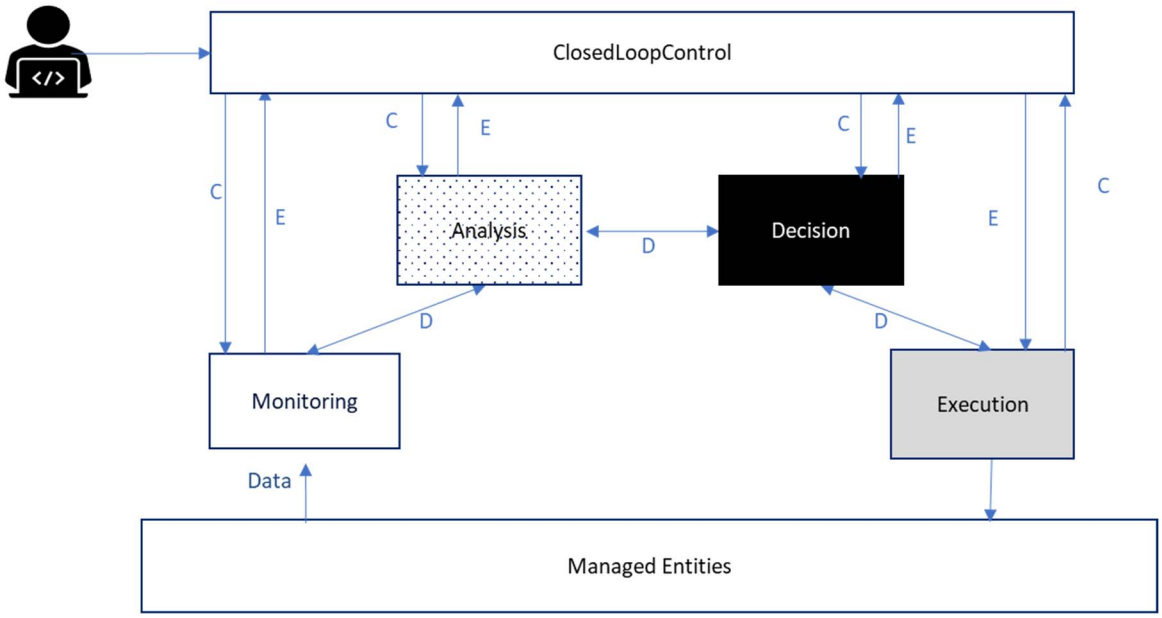

Figure 5.3-1: Automatic composition and management of multi-vendor closed loop

Given the likely high number of CLs and combinations thereof, it would be a heavy task for the human operators to manage. Hence It is necessary to provide automation means with which CL stages may be dynamically composed.

Challenges:

Made-to-Order Closed Loops (M2O-CL), as defined in clause 7.4 of ETSI GS ZSM 009-1 [i.2] and service capabilities further described in clause 5.4.6 of ETSI GS ZSM 009-2 [i.3], are assembled on demand by ZSM framework owner, or by other entities on behalf of the ZSM framework owner, using capabilities offered by the ZSM framework. Components of M2O-CLs may come from different ZSM framework vendors and can be associated with a CL instance based on demand. As described in ETSI GS ZSM 009-2 [i.3], it defines the capabilities needed to chain the different components together to prepare a M2O-CL for deployment.

The composition process - according to current solution - does not support means to request for the dynamic composition of the closed loop. Accordingly, such means do not reflect sufficient flexibility to automatically decide the number and type of needed stages and how to handle situations where one or more needed components are not available. Moreover, a flexible and abstract way to submit a M2O-CL composition request needs to be supported.

To provide a flexible/abstract way to submit a M2O-CLs composition it is important.

- To determine the minimum needed information to form M2O-CLs. The needed information may be:
  - Catalogue(s) of CL components
  - CL goal and description
  - Required capabilities of management functions that combine to form M2O-CLs
- To determine the required number of stages and capability to form M2O-CLs.
- To configure the M2O-CL stages, e.g. with their sources of data or where to deliver their reports.
- To map output of one stage to the input of subsequent stages e.g. data transformation might be required between the stages to facilitate mapping.

5.4 Intent-based closed loops

Intelligent networks need to have the ability to adapt to new situations and new services without the intervention of human operators or experts. One needed feature that is necessary for any intelligent network is to decouple the requirements that are given to the management systems from the solutions that are used to deploy and manage the services. As defined in ETSI GR ZSM 011 [i.4], intents are the formal definition of the requirements, goals, and constraints that a technical system receive and should be used as the main input information for autonomous management of the network.

An intent is a declarative information object that allows the system to understand utility and the value of its actions from a producer's perspective. This allows the system to become autonomous and evaluate situations and potential action strategies, rather than being just limited to follow instructions that operators and experts have specified in policies. This means that intelligent decisions that were previously made exclusively by policy developers can become automated.

On one hand, intents are necessary for providing information about requirements and utility, which is one of the foundations for autonomous systems; on the other hand, closed loops are the key enablers for automating the configuration of networks and steering their state towards the wanted state. The combination of these two concepts, i.e. intents and closed loops - which is needed to realize intelligent and autonomous networks - is still a challenging task.

As described in ETSI GR ZSM 011 [i.4], the management domains may contain Intent Management Entities (IME), which are responsible for managing the life cycle of intents. This implies that an IME is able to detect whether the intent has been fulfilled and, if not, find and execute corrective actions; therefore, IMEs have to leverage closed loops functionalities while dealing with the intent lifecycle management.

Figure 5.4-1 shows the main interactions between IMEs within different management domains (which may include the end-to-end management domain). The IME receives all intents directed toward its autonomous domain and it has to report back to the intent origin regarding its success in fulfilling the intent. The intent fulfilment has to be executed based on measurements that allow the IME to compare the state of the network to the expected state derived from the intent. If the system is not meeting the requirements, corrective actions are needed. The actions can be through conventional management interfaces or, if the targeted is intent-aware, by defining its requirements through another intent.

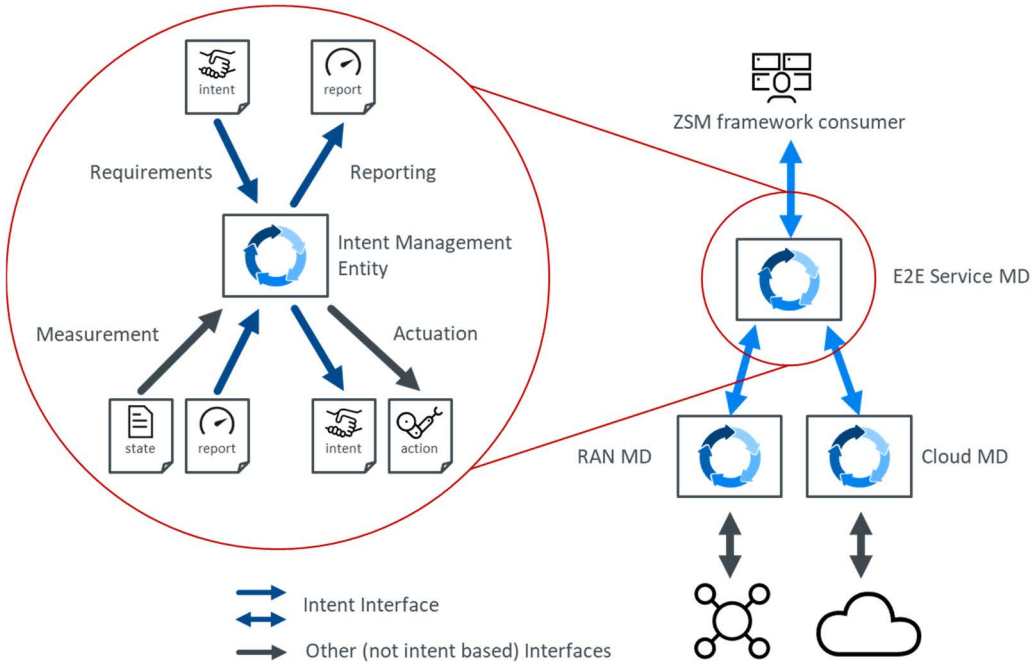

Figure 5.4-1: Automatic composition and management of multi-vendor closed loop

Challenges:

Intent-based operations are well explained in ETSI GR ZSM 011 [i.4] and in other standards ([i.5] and [i.10]).

As investigated in ETSI GR ZSM 011 [i.4], the IMEs operate different closed loops that are responsible to produce and maintain the outcomes expected from the intent received, e.g. a service with specified QoE.

The implementation of IMEs and the use of closed loops can be realized in different forms, but a general challenge is how the IMEs interact with the specific business logic within a management domain, so that they can together translate an intent that expresses business needs into detailed technical configurations.

Such translation can be done in generic way, such closed loop enablers can be used to create a generic closed loop structure that is able to deal with any type of intent, given they follow a common syntax and modelling. This would relieve the burden of implementers that would not need to design and code specific (and different) closed loops for each type of intent that is employed.

The challenge that should be tackled is to create intent-driven closed loops that have a generic structure and that:

i) are able to deal with any type of intent, and

ii) can be specialized to translate business logic into technical configurations in any management domain.

5.5 Resource locality and scarcity on closed loop automation

The implementation of a closed loop may require a certain number of resources. The number of resources may potentially grow with the complexity of the activities expected from the closed loops. Complex operations that are likely to be performed by a closed loop might include artificial intelligence (AI) model training, data analysis and other non-trivial decision-making processes. At the same time, some resources that are supposed to be managed by closed loops may be hosted in local nodes, quite far (and in some cases, isolated) from central nodes. This is the situation that industry foresees with the on-going trend of IoT-edge-cloud continuum, where managed resources do not only include only the ones hosted in centralized and regional data centers, but also on extreme-edge nodes (also known to as the far-edge).

As for the management of extreme-edge node resources, there exists different variants.

On the one hand, implementing the closed loop locally, on the extreme-edge node; this means the closed loop and the resources under its managed scope will both run on the same execution environment. The main issue of going for this approach is that it may be very costly (per compute unit), or ineffective, as it would deplete a large part of the scarce resources available in the extreme-edge node.

On the other hand, implementing the closed loop remotely, on a central node. However, this approach might pose serious issues related to the exchange of data and control flows between the extreme-edge and central nodes. Examples of these issues can include:

- disconnection risks between both nodes, which may result in non-acceptable service downtime;
- data sharing: moving local data from extreme-edge node to central node for their processing may raise privacy concerns, especially when these data hold sensitive (business-critical or regulation-subjected) data;
- performance degradation, in terms of latency (e.g. increased time to send the monitoring data and receive action plans) and backhaul capacity (more exchange of data and control flows between nodes mean higher bandwidth consumption, which might result in poor resource efficiency and congestion).

Challenges:

Currently, resource locality is not taken into account when considering closed loop instantiation and operation. While cognitive and self-learning closed loops are expected to bring better results than traditional non-self-learning ones, they also require larger amount of data and processing power to operate, which is not necessarily compatible with specific use cases where local resources are scarce, such as mobile network edge resource management.

Centralizing the closed loop in an area with more resource may result in increased risk of disconnection from the managed resources. In many cases, such as security, such disconnection may not be acceptable.

Similarly, centralization may induce additional delays in the loop, which may also be a problem if the action plan needs to be carried out in a timely manner.

6 Potential solutions

6.1 Cognitive Closed Loops

6.1.1 Active measurements

As discussed in clause 5.1, a CL may host multiple ML models to enable a wide range of analytics capabilities. Different models usually require different type and amount of input data to produce their inference result. For example, a model with few input data may be able to deliver coarse analytics results, while another model with more diverse input data may deliver finer analytics results. The two models may be different in complexity and inference speed, i.e. the simpler model may be quicker for basic decisions. Additionally, the simpler model is more relaxed on the data collection requirements, i.e. puts lower load on the data producers (e.g. other CLs or managed entities). Therefore, the mode of operation of the CL could be to normally use the simpler model driven by a limited set of data, whereas it proceeds to use the more complex (and more data intensive) model only in exceptional cases when the simpler model's inference is not satisfactory or conclusive.

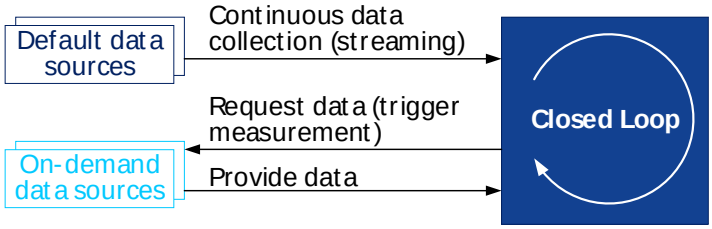

Figure 6.1.1-1: Dynamic interaction with data sources

Optimizing data collection according to the above mode of operation requires that the CL be able to dynamically select data sources according to its internal analytics flow. Accordingly, the CL may use a set of "default data sources" whose data are collected and analysed when the CL operates under in normal conditions, while additional "on-demand data sources" are utilized only on need basis such as when the CL faces unusual or abnormal conditions, and thus such additional data are collected only on specific request from the CL. Such operation is outlined in figure 6.1.1-1.

The on-demand data collected on need basis may be used in combination with already available historical data (figure 6.1.1-2(a), when a model has been trained on a combination of regular and on-demand data) or used separately (figure 6.1.1-2(b), when a model is trained solely on the on-demand data).

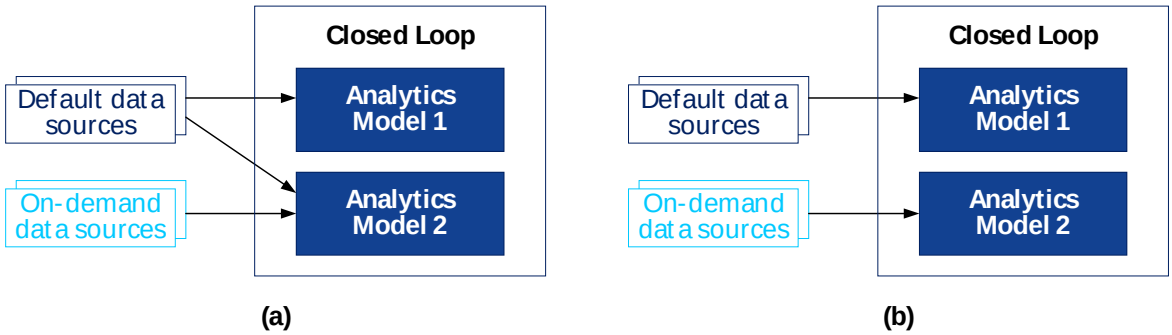

Figure 6.1.1-2: Closed Loop with multiple analytics models, each requiring different (potentially overlapping) set of input data

On-demand data may have at least two availability stages:

1) Pre-available/existing data. This is data that has already been produced by a data source and has been stored in a database, from where it only needs to be retrieved.

2) Non pre-available/non-pre-existing data. Such data needs to be generated on request. This data has not been produced prior to the request and requires additional actions/procedures (outside of the CL) to be generated.

Requesting existing data is usually more lightweight and available with shorter latency compared to the data that needs to be generated, especially if the latter implies interactions between network functions or measuring/generating user plane traffic. Still, on-demand generated data is useful to enable a CL to perform more detailed analysis; and in some situations, the availability of pre-existing data may not be always guaranteed (e.g. due to deployment or storage constraints).

With certain ML technology, the CL may detect that a model is not able to deliver conclusive or confident results by evaluating the module's intrinsic confidence metric, such as that of a SoftMax classifier used as the output layer of a deep neural network. Alternatively, the CL may monitor the distribution of model input data encountered on the field and compare it to the current default model's training data distribution to check if the model is dealing with the kind of data, it has not been trained for, triggering a switch-over to another model that was trained for the currently received data. Other analytics specific means may also exist that the CL may use to realize that a given model is not sufficient for its analytics/decision step and another model, therefore additional data is required.

The flow of command where a CL triggers active measurement as part of its analytics/decision steps is shown in figure 6.1.1-3. The CL is depicted by its multiple internal stages (Collection, Analytics, Decision, Actuation). The Analytics stage realizes (by means discussed above) that the default collected data does not lead to conclusive results, therefore it decides to trigger a new measurement. The new measurement is shown to be initiated by the CL's Actuation step, emphasizing that obtaining the on-demand data may require new data to be generated by entities outside of the CL, not only fetching additional (but already existing) data from a database.

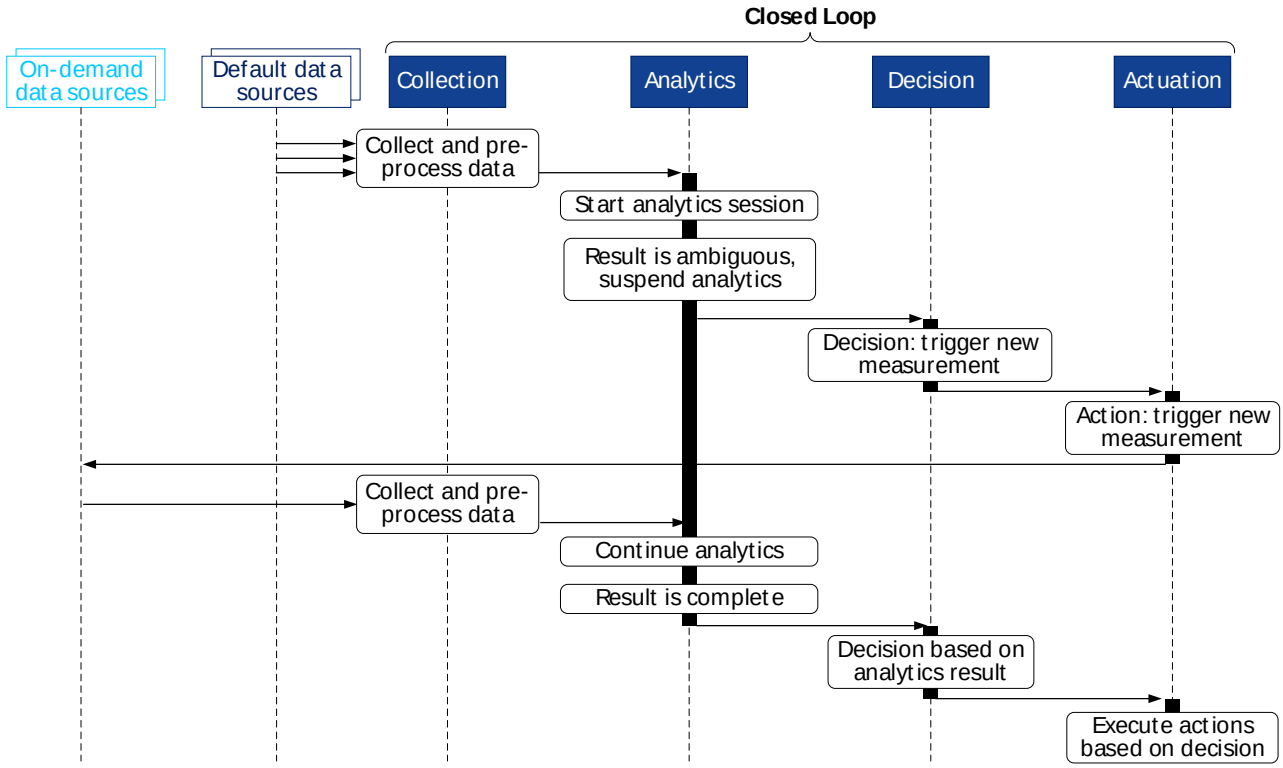

Figure 6.1.1-3: Procedure for triggering active measurements from a Closed Loop

6.1.2 Self-learning closed loops

6.1.2.1 Introduction

Closed loops autonomously trigger actions based on a series of steps including data collection, analytics, and decision. During a CL's decision step, the closed loop may select from a set of actions, where the best action depends on the CL's dynamically changing environment and operational context. In systems with dynamicity (such as mobile networks with mobility, traffic diversity, etc.), the mapping between potential contexts (such as domain specific insights and system states generated by domain intelligence) and actions cannot be defined statically (e.g. as a list of rules) prior to the implementation and deployment of the CL. Therefore, CLs are expected to autonomously adapt their operation to varying environments, collect operational knowledge and autonomously learn from their experience.

Such self-learning capability complements external CL supervision when an entity outside of the CL evaluates the performance of a CL and may adapt the CL's operation.

6.1.2.2 Evaluation of the efficiency and impact of closed loops

Self-learning requires that CLs evaluate their decisions and actions in the context in which they were initiated. CLs may generate self-learning feedback from their own actions by correlating the context in which the actions were initiated, the action itself, and measuring their efficiency and impact the action had on the entities managed by the CL. The context, efficiency and impact terms are defined as follows.

Context: the context may be derived from the CL's collected data (historical and current), any past knowledge and internal insights created by its analytics step. The context may not only be a snapshot of the state of the managed entities at the time of initiating the action but could (and often should) include longer term historical and temporal behaviour of the system, as well as the other actions that have been in effect.

Efficiency: describes whether the action has been successfully completed with the specific scope and target (UEs, cells, etc.). An action may be partly inefficient if its outcome depends on conditions that are context specific, such as traffic, mobility, UE capability, service availability or any other dynamically changing state. Therefore, a CL may store the actions, the context in which the action was triggered and the measured efficiency of the action to learn what actions are efficient in a given context and filter out actions from contexts where they consistently prove to be inefficient.

Impact: describes whether the action has reached its goal, which should be aligned with the CL's operational target (e.g. resolve a system state degradation). Even an efficiently executed action may have low impact if the action was not the proper one to resolve the motivation of the action. Therefore, a CL may store the measured impact of the actions along with the efficiency to filter out low impact actions. Note that an efficient action may have various impact based on the force of the action, i.e. the same action may be administered lightly or more strongly based on parameterization. Low impact of an otherwise efficient action does not necessarily mean that the action itself was wrong, it may be that the action would have needed different parameters or strength. Therefore, the CL should also note the parameters of the action to enable tuning the actions similarly to control-loop tuning in control theory.

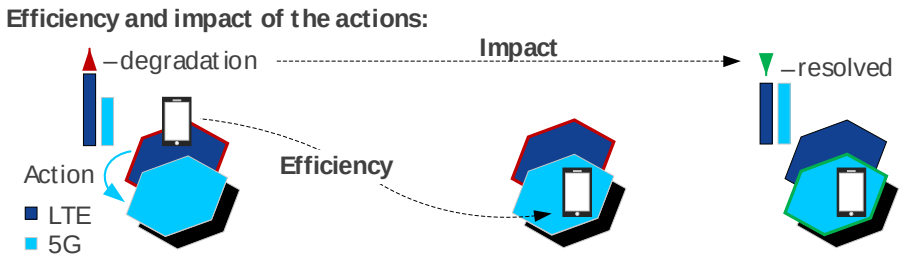

Figure 6.1.2.2-1: Example scenario for measuring the efficiency and impact of an action

For illustration, an example scenario is depicted in figure 6.1.2.2-1. The scenario consists of two resource layers, LTE and 5G, where the LTE layer is overloaded causing service degradation. The action triggered by the CL is traffic steering (TS) to move selected UEs from LTE to 5G. The efficiency of the action is measured by evaluating how many of the UEs targeted by the TS action have indeed moved to 5G. Efficiency below 100 % may be caused by, e.g. inadequate 5G coverage at their exact location. The impact of the action is measured by evaluating whether the original cause of the action, i.e. the degradation in LTE, has been resolved. It may be that the UEs handed over to the 5G are moving sufficient load away from LTE and onto 5G that the LTE degradation is resolved, which would mean high impact (good case). A low impact case would happen if, despite moving UEs from LTE to 5G, the LTE situation remains poor. Additionally, low impact should be declared if, due to moving load from LTE to 5G, the 5G layer becomes degraded. In general, the impact should not only consider whether the original cause has been resolved, but also whether the action has caused additional issues - those should be avoided as well for an impactful action.

6.1.2.3 Timing aspects

Actions may have different latency until they become effective (i.e. until their outcome becomes measurable), therefore even an efficiently completed action may not deliver the expected impact in time. Therefore, considering the timing aspect of the actions is essential for making efficient and impactful decisions by the CL. In general, if the action is triggered as a remedy to a degradation, the action should complete (take effect) before the issue escalates to degradations noticeable to end users or SLA violations.

Therefore, the CL should measure and store the latency of the actions it triggers, and in any situation select actions that have a chance to take effect within the time budget available to resolve the trigger condition.

The timing of actions is also relevant for tuning the action (cf. the earlier control-loop analogy), as tuning an action's parameters requires knowledge on the time constant of the action (i.e. the speed at which changing the action parameters is expected to take effect in the action's observed efficiency and impact).

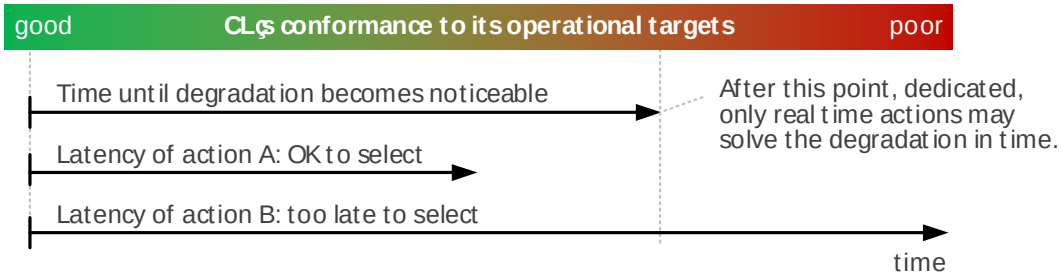

Figure 6.1.2.3-1: Illustration of the concept of considering the latency of actions in CL analytics & decision making

The concept is illustrated in figure 6.1.2.3-1 with a situation where, with time, the system would transition into a degraded state, which is monitored by the CL. The objective of the CL is to keep the system out of the poor performance range. Suppose that in the current context, there are two actions that have proven to be efficient and impactful for the CL, indicated by "A" and "B". The latency profiled for action "A" is shorter than the time before the degradation is expected to escalate, making "A" a potential choice for the CL. However, the latency of action "B" has been measured to be longer than the time until degradation, therefore, even if action "B" was an efficient and impactful one, in the current situation it should not be selected if there is a better alternative. Therefore, the CL should go with action "A" in this case. Action "B" is still a valuable tool as a long-term action that is however impactful only in case of slow system changes or as proactive actions.

6.1.2.4 Self-learning CL operation

The self-evaluating and self-learning CL operation is depicted in figure 6.1.2.4-1, combining the concepts of efficiency and impact evaluation as well as latency considerations, and mapping them onto the CL steps of data collection and analytics, decision, actuation. The contextual knowledge representation is also highlighted.

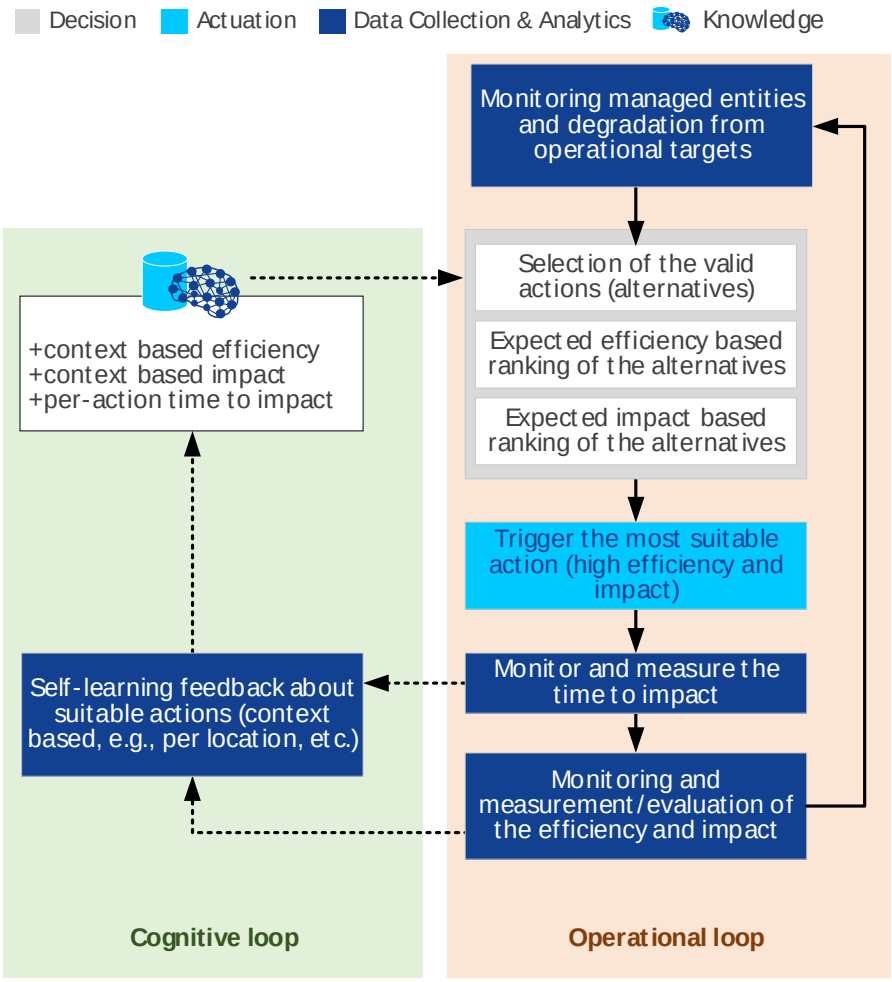

Figure 6.1.2.4-1: Self-evaluating and self-learning CL operation

The CL may be logically partitioned into two interacting internal loops: a Cognitive Loop and an Operational Loop. The Cognitive Loop is responsible for maintaining the context-based knowledge of the CL, based on self-monitoring the actions and their efficiency, impact and latency. The Operational Loop has the scope to collect data from the managed entities and observe the context in which the CL operates; detect if there is any need for an action (degradation from CL operational targets); select, based on the accumulated knowledge, the best possible action in the context; launch the selected action; monitor the efficiency, impact, and latency of the action. The interaction between the two logical loops is to generate self-learning feedback from the Operational Loop for the Cognitive Loop, and to retrieve knowledge from the Cognitive Loop to the Operational Loop.

Note that at a given decision point, prior accumulated knowledge may be insufficient to support the decision of the CL about triggering actions. In this case, the CL may transition to an exploration mode, where actions are evaluated based on pre-defined rules or general (not context specific) domain knowledge used for bootstrapping the CL. In theory, random action selection as in reinforcement learning exploration may also be used, however this may not be suitable in certain CL deployment environments (e.g. business critical deployments). In such cases, domain knowledge may be used to limit the explorable actions (or the extent of the actions) to a safe range.

Another possibility to boost learning is to leverage the collective knowledge of multiple CL instances that implement the same business logic but that are deployed at various locations (e.g. at different radio access nodes). In such cases, combining knowledge accumulated from multiple CLs and feeding it back to CL instances has advantage over isolated self-learning as multiple CLs may cumulatively encounter richer context and exhibit experience of more diverse actions. Additionally, bootstrapping the knowledge of a newly deployed CL with a combined knowledge may fast-track the self-learning of the individual CL as well.

6.2 Composition and Coordination of interdependent Closed Loops

6.2.1 Grouping interdependent (nested) Closed Loops

As discussed in clause 5, multiple instantiated CLs belonging to the same use case with a common goal or related goals may act directly or indirectly on the same managed entity. Therefore, new solutions are required to compose and coordinate such interdependent CLs to ensure that their individual predicted actions do not conflict with each other resulting in an undesired operation on the managed entity.

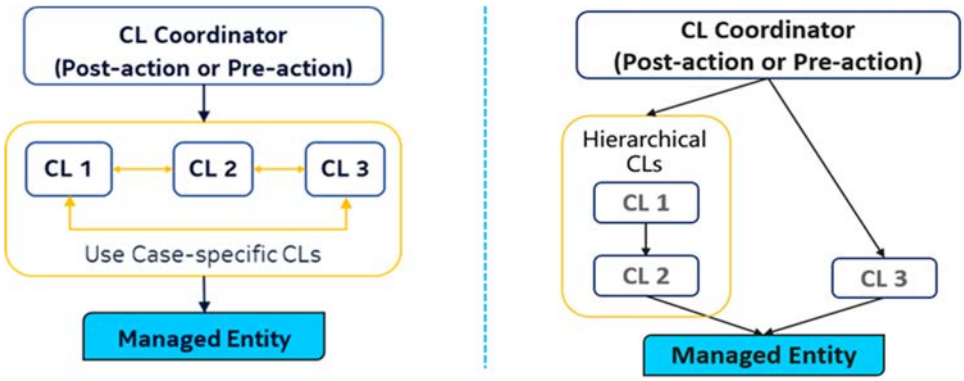

Figure 6.2.1-1: Interdependent (nested) Closed loops

CLs belonging to the same use case with a common goal or related goals may be composed as a *group* of interdependent (nested) CLs. These CLs may coordinate themselves in a hierarchical manner, without the need for an external CL Coordinator, to detect and mitigate conflicting actions. The grouped interdependent (nested) CLs may appear as a single entity to the external CL coordinator, as shown in figure 6.2.1-1.

The grouped interdependent (nested including hierarchical) CL model may be defined by introducing new attributes (as shown in table 6.2.1-1) in addition to the attributes existing in CL model that was defined in ETSI ZSM009-1. In the case of hierarchical CLs, subordinate CL can be defined as manageableEntityList of superior CL (as defined in ETSI GS ZSM 009-1 [i.2], clause 8.1.4.4).

Table 6.2.1-1: Attributes for Interdependent (nested) closed loops

Attribute Name	Optional / Mandatory	Description
closedLoopGroupInstanceUniqueId	Mandatory	It indicates the identifier of the grouped CL instance.
closedLoopGroupGoal	Mandatory	It indicates the common goals of the grouped CL instance.
closedLoopGroupPriority	Mandatory	It indicates the priority of the grouped CL instance. It is set to avoid conflicting actions (i.e. between two or more grouped CL instances) to the same managed entity.
closedLoopGroupCoordinationType	Mandatory	It indicates the type of hierarchical coordination approach (i.e. hierarchy either increases or decreases from the innermost CL to the outermost CL) within the grouped interdependent CL instance.
closedLoopGroupTypeDescription	Optional	It indicates a description of the grouped CL type (Made-to-Order CL or Ready-Made CL).
closedLoopClass 1 (Innermost CL)	Mandatory	It is of type closedLoopClass and defines the attributes of the Innermost CL.
closedLoopClass 2 (Intermediate CL)	Mandatory	It is of type closedLoopClass and defines the attributes of the Intermediate CL. Note that there can be 0, 1 or many Intermediate CL instances.
closedLoopClass n (Outermost CL)	Mandatory	It is of type closedLoopClass and defines the attributes of the Outermost CL.

6.2.2 Coordination of interdependent (nested) Closed Loops

The grouped interdependent (nested) CLs may coordinate with each other to ensure that their individual predicted actions do not conflict with each other. Such coordination mechanism ensures that at a time only the Decision stage of one of the interdependent CLs will send their insights & workflows to the corresponding Execution stage (i.e. the Decision stage of other CLs will be in pause mode).

Solution 1

As shown in figure 6.2.2-1, the insights & workflows from the Decision stage of *one* CL are fed as an input to the Analytics stage of next (or all) lower or higher CL(s). The Analytics stage of one CL may control or configure (e.g. enabling or disabling pause points) the Decision stage of other (or all) lower or higher CL(s).

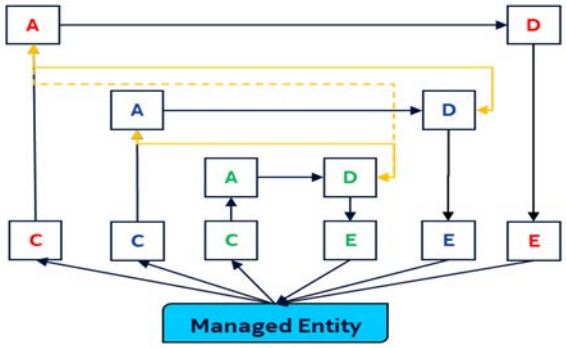

Figure 6.2.2-1: Coordination of Interdependent Closed Loops - Solution 1

Solution 2

As shown in figure 6.2.2-2, if Analytics and Decision stages of a CL are combined into a single Analytics & Decision stage, then the insights & workflows from the Analytics & Decision stage of one CL are fed as an input to the Analytics & Decision stage of the next (or all) lower or higher CL(s). The "Analytics & Decision" stage of one CL may control or configure (e.g. enabling or disabling pause points) the "Analytics & Decision" stage of the next (or all) lower or higher CL(s).

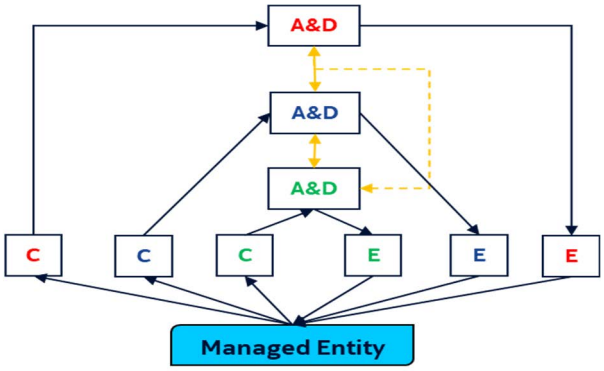

Figure 6.2.2-2: Coordination of Interdependent Closed Loops - Solution 2

6.2.3 Example Workflows for Coordination of interdependent (nested) Closed-Loops

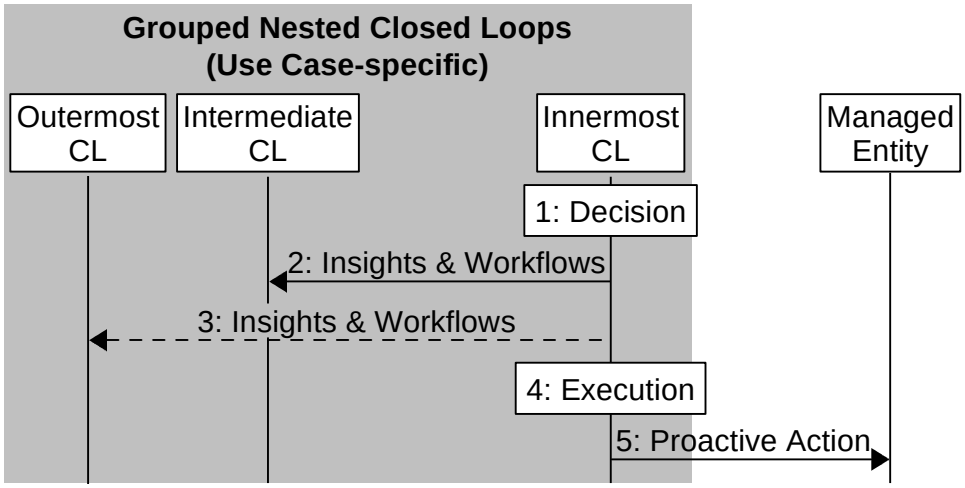

Figure 6.2.3-1: Workflows for Coordination of interdependent closed loops

Table 6.2.3-1: Capabilities required for interdependent closed loops

Insight Reporting & Control (One CL Decision Stage -> Another CL Analytics Stage)	
Capabilities	**Description**
Transfer insights & workflows report (raw or aggregate) from one CL to another CL	Transfer a file or stream data on the derived insights & workflows which need to include at least the following information: • Specifies the scope of the target managed entity (e.g. CPU, VNF, node) that the CL is acting on • Individual goal of the CL (e.g. maintain sufficient CPU resources in the VNF depending on the traffic load to avoid service degradation) • The predicted decision/action (e.g. add/remove one/more CPU units in the VNF) • Accuracy/confidence of the prediction (e.g. x % accurate with confidence value y) • Timestamp when the predicted action is planned to be executed (e.g. t+5 minutes)
Escalate/signal problem about the CL to another CL	Transfer a file or stream data on the performance of the CL. For e.g. indicating that the required data to train an ML algorithm is not available or indicating that the accuracy/confidence of the prediction is poor or indicating that by performing the predicted action the value of the *closedLoopGroupGoal* attribute might not be achieved.

In figure 6.2.3-1:

Example 1: The Decision stage of a CL in the CL Group may derive insights & workflows by applying analytics (e.g. Machine Learning algorithms) on the data collected in its Collection and Knowledge stages of the CL.

Example 2-3: The derived insights & workflows (e.g. predicted/proposed actions) from one CL may be sent to the Analytics stage of the next (or all) CL(s) via the Insight Reporting & Control Interface.

Example 4-5: The derived insights & workflows are received by the Execution stage of the CL that derived the insights & workflows, which triggers the proactive action towards the managed entity.

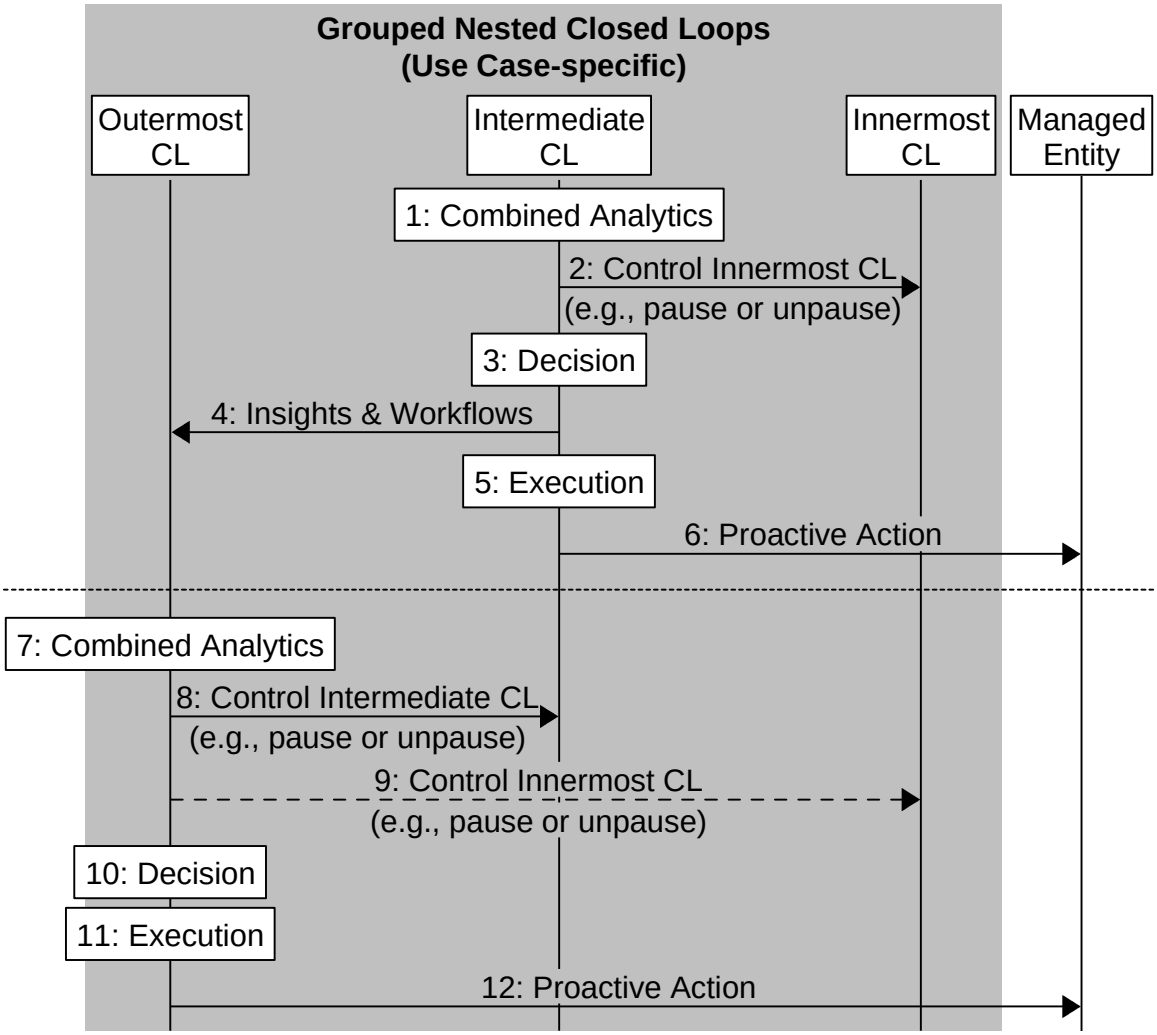

Figure 6.2.3-2: More workflows Coordination for interdependent closed loops

Table 6.2.3-2: More Capabilities for interdepended closed loops

Insight Reporting & Control (One CL Analytics Stage -> Another CL Decision Stage)	
Capabilities	**Description**
Pause/unpause another CL(s)	Set pause point on the Decision stage of another CL(s) by indicating how long the CL needs to be in pause state. Alternatively, the CL may be unpaused only when the action/trigger is requested by other CL(s) in the *closedLoopGroup* instance.
Update/reconfigure individual goal of another CL	Updates/reconfigures the value of the *closedLoopGoal* attribute of another CL instance.
Configure insight & workflow report requirements to be received from another CL	Configure whether raw reports are needed or aggregated reports (also specifying the time granularity) is sufficient for performing analysis.

In figure 6.2.3-2:

Example 1 or 7: *A CL within the CL group could combine the analytics on data collected by itself and the insights & workflows from the decision stage of the other CLs within the group to determine the best course of action.*

Example 2 or Example 8 or Example 9: The control message (e.g. pause) is sent by the Analytics stage of one CL (e.g. intermediate) via the Insight Reporting & Control Interface to the Decision stage of the next (or all CL(s)).

Example 3 or Example 10: The Decision stage of one CL (e.g. intermediate) may derive insights & workflows by applying analytics (e.g. Machine Learning algorithms) on the data collected in the Collection and Knowledge stages of the same CL.

Example 5-6 or Example 11-12: The Execution stage of one CL (e.g. intermediate) triggers the proactive action towards the managed entity.

6.2.4 Example Workflows for Coordination of interdependent (hierarchical) Closed-Loops

As ETSI GS ZSM 009-1 [i.2], clause 9.2.4 described, the Closed Loops execution management service provides the ability to pause the execution of a Closed Loop at a "pause point". In the case of pausing the hierarchical Closed Loops, there may be two possibilities to consider:

1) In most cases, when the superior CL is paused, the subordinate CL is paused as well.

2) If goals, actions, etc. in different CLs are independent of each other, the coordination between CLs may not always happen. In this case, when the superior CL is paused, the subordinate CL continues working.

The possible way to deal with above possibilities is providing instruction to describe what the subordinate CL should be done when superior CL is paused/resumed.

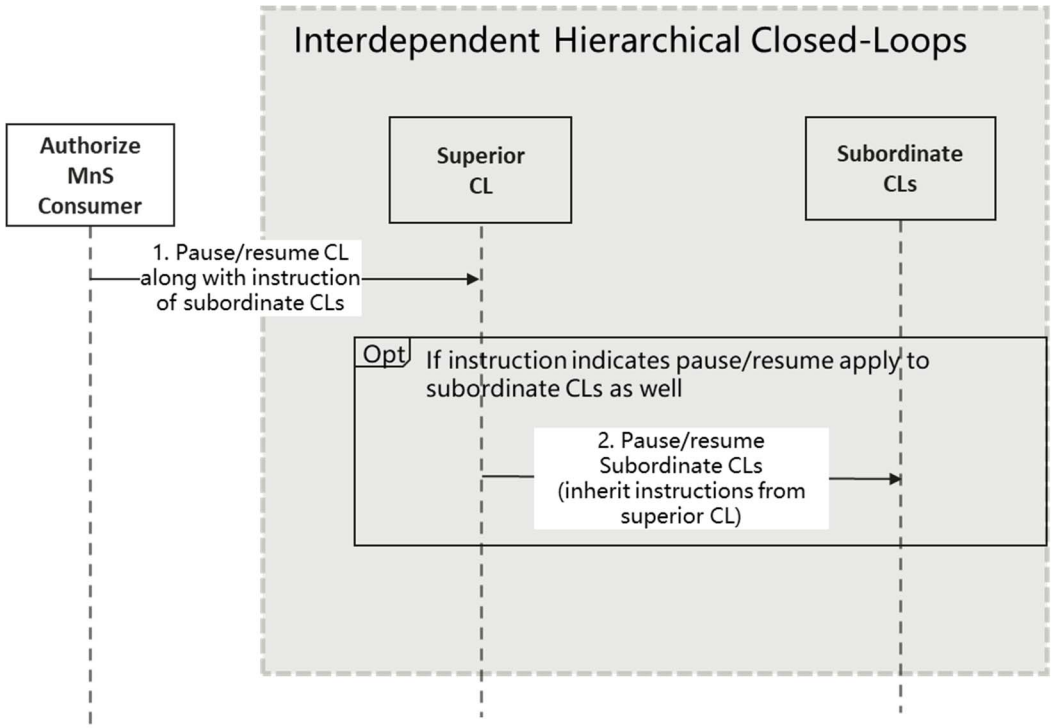

Figure 6.2.4-1: Workflows of Consumer-triggered interdependent closed loops Coordination

Table 6.2.4-1: Capabilities required for interdependent closed loops

Control (Consumer -> CL Coordination)	
capabilities	**Description**
Configure instruction whether to perform the same pause/resume operation on the subordinate CLs when pause/resume a CL.	Pause/resume a closed loop with subordinate CLs operation instruction. • When superior CL will be paused at the next reachable pause point, the instruction can indicate whether subordinate CLs are paused as well. • When superior CL is resumed, the instruction can indicate whether pass the resume operation to subordinate CLs.

In figure 6.2.4-1:

Message Flow 1: The pause/resume message sent by an authorized consumer may carry an instruction of what the subordinate CLs should be done upon the action of the superior CL.

Message Flow 2: If message flow 1 acts on a CL with the instruction indicates that subordinate CLs are paused/resumed as well. The superior CL sends pause/resume message to each subordinate CL.

NOTE: According to definition in ETSI GS ZSM 009-1 [i.2], clause 9.2.4, the Closed Loops execution management service can be a producer to receive the pause/resume message using the "Pause a Closed Loop" or "Continue Closed Loop execution" capability, and to provide the notification to authorized consumer as required using the "Provide notification for a pause point reached" capability.

6.3 Dynamic Composition of Closed Loops

To provide enablers for automation, one may apply a Closed-Loop Controller (CLC), described in clause 5.3, which is responsible for composing the closed loop based on the input "context" (e.g. capabilities, control) provided by the ZSM framework consumers. The CLC replaces the ZSM framework consumers for composing the closed loop using information about CL components and stages.

NOTE: As described in ETSI GS ZSM 009-1 [i.2], CL stage is the logical role to be played while the CL component is the entity identified to play the particular role(s).

To automate the composition of the closed loop:

1) The ZSM framework consumer sends a request to the CLC to compose a CL. The request may specify inputs and information about the CL components or roles/stages that are needed to compose the CL. Such information may include:

   - CL-component identifiers: The ZSM framework consumer may know (via pre-arrangement per use case, capabilities, and roles) the exact components which need to be composed into a CL. Then, the ZSM framework consumer identifies the components using their unique names or identifiers such as Calculate_mean or Statistical_analysis and defines their roles/stage (e.g. Decision, analysis, etc.). Subsequently, the ZSM framework consumer provides CL component identifiers and their roles as the input to the composition request sent to the CLC.

   - CL-component inputs and outputs: The ZSM framework consumers may not know the exact components (their names or identifiers) that are needed for the different steps of the CL composition but may know the inputs needed to be used and outputs to be generated at different steps in the CL. In that case, the ZSM framework consumers identifies the tuple of inputs and outputs for each CL step and labels it with the respective roles for that tuple. Subsequently, the ZSM framework consumers provides CL components roles (e.g. one more CL stages) and inputs-outputs tuples as the input to the composition request sent to the CLC.

   - CL-Step capabilities: Alternatively, the ZSM framework consumer may not know the inputs and outputs required at different steps in the CL but may know the descriptions of the capabilities (e.g. analysis of handover performance) of the corresponding components (or a subset of them). In that case, the ZSM framework consumer describes the expected capabilities required for each CL component and the respective role for that set of capabilities. Subsequently, the ZSM framework consumer provides the CL component roles and capability description as the input to the composition request sent to the CLC.

   - Combinations of CL-Step capabilities, inputs, and outputs: The ZSM framework consumer may also state the combination of capabilities with inputs, capabilities with outputs or of capabilities with both input and outputs as the basis for composing the CL.

2) The request sent by the ZSM framework consumer triggers the composition process. Thereby, the CLC identifies and finds the required components as described by the ZSM framework consumer's request i.e. in terms of the components' unique names or identifiers, components' inputs and outputs tuples and/or components' capabilities.

3) The CLC composes the CL by configuring the relation and data exchange among the individual identified components. This may be achieved using the services define in clause 5.4.6 of ETSI GS ZSM 009-2 [i.3] for the Made-to-Order Closed Loops (M2O-CL). Some examples are:

   - The CLC may configure the monitoring component to deliver data to the analytics component.

   - Alternatively, The CLC may configure the analytics component to collect data from the monitoring components whenever such data is needed to compute analytics.

- The CLC may configure the analytics component to deliver data to the Decision component.
- Alternatively, the CLC may configure the Decision component to request and collect data in the form of an analytics report from the analytics component whenever such data is needed to derive a decision.
- The CLC may configure the Decision component to deliver data in form of recommended actions or commands to the Execution component.
- Alternatively, the CLC may configure the Execution component to collect data in form of a set of recommended actions from the Decision component whenever such recommendations are needed.
- The CLC configures the Execution component to implement the recommended actions on to the network once received or upon certain conditions, e.g. threshold surpass.
- The CLC may also configure the components to provide Performance Management (PM) and/or Key Performance Indicator (KPI) and/or feedback data on their operations.

6.4 Intent-based closed loops

6.4.1 Intent-driven closed loops

Figure 6.4.1-1 illustrates how a closed loop could be implemented within an IME to execute the intent fulfilment; this is an example of an intent-driven closed loop. This closed loop consists of logical stages that are implemented by different types of agents: Measurement, Issues, Solutions, Evaluation and Actuation. Compared to the ETSI GS ZSM 009-1 [i.2] functional representation of closed loops, Measurement agents implement the Monitoring stage; Issues agents implement the Analysis stage, Solutions and Evaluation agents implement the Decision stage, while Actuation agents implement the Execution stage [i.2]. Each logical stage is implemented by a software agent [i.6] that is specialized in each task of the closed loop execution; different agents can be coded to perform the same task and different agents can be used depending on the conditions of the environment or the performance of the closed loop execution.

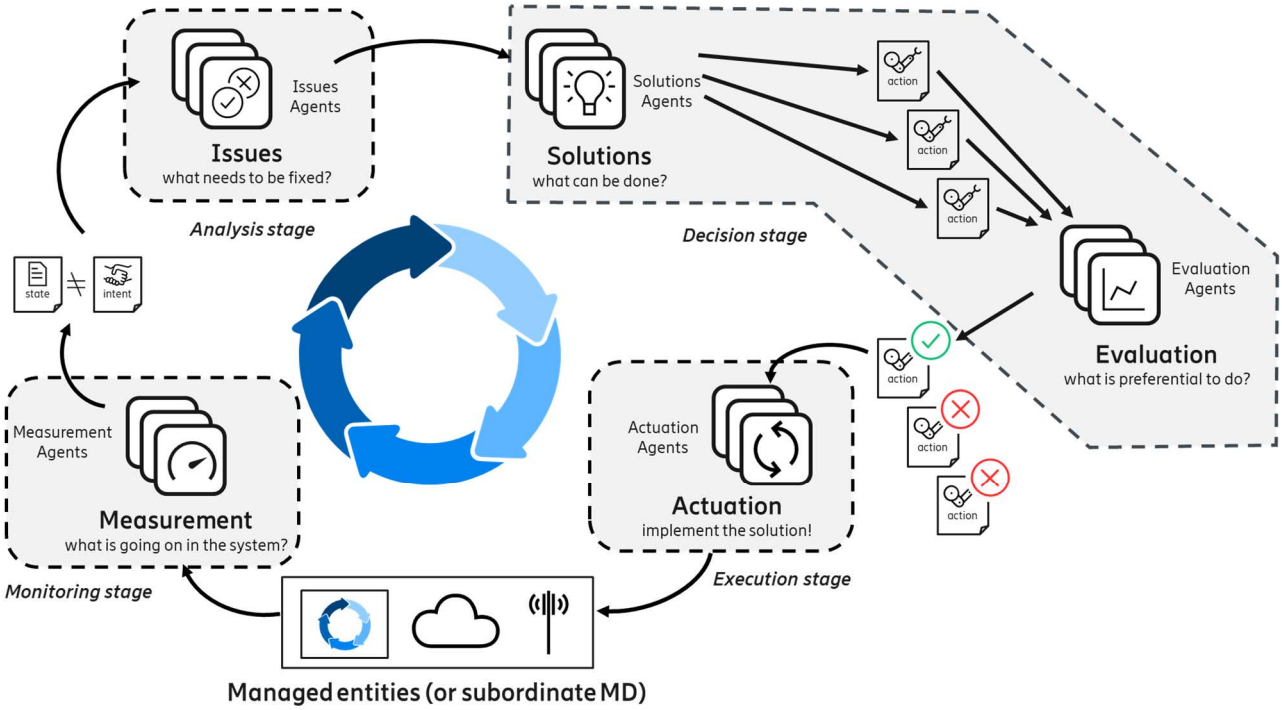

Figure 6.4.1-1: More workflows Coordination for interdependent closed loops

Measurement is a continuous activity used both to set goals and to evaluate the intent fulfillment. The IME needs to know which resource instances are used to fulfill the intent as well as their operational state. Knowledge of the current performance of service components at their resource instances makes it possible to determine if the system is in breach of its intent and needs corrective action. However, as the intent can change, so can the required measurements. The IME should adapt its measurements accordingly by employing measurement agents (specialist implementations of measurement tasks) as required.

The Issues stage focuses on identifying what needs to be fixed. Issues are defined as requirements that are not met by the system. Examples include a required QoE metric that is not reached or a required resource that is not available. Assurance agents implement continuous monitoring of state versus requirements and raise issues when required. This can include a root cause analysis for a more precise description of the issue, prioritization of issues depending on their severity or waiting for ongoing actions showing results.

In the Solutions stage, proposal agents react to issues and determine the available corrective action strategies. Any number of differently implemented proposal agents can be used at this stage, including human designed policies and machine learned policies that derive solutions from evidence data. Multiple action strategies may be proposed, each of them representing what the system can do to address the issue.

In the Evaluation stage, evaluation agents determine the expected impact of an action on the system state to determine which of the proposed solutions can deliver the most positive effect on the fulfillment of all currently valid intents. Predictive models and digital twins will play a major role in this step, as they make it possible to virtually explore actions and their expected outcomes. The preferred action proposal is the one expected to maximize global utility by best fulfilling all intents. The Evaluation stage can also include detection of conflict from actions that fulfill one intent at the expense of degrading another. Evaluation can therefore be a sanity mechanism with the potential to save the network from risky actions and degradations.

A solution with an overall preferential evaluation proceeds to the Actuation stage. Intent-based systems can act by using an intent to define requirements on the autonomous subordinate system layer. Alternatively, they may act through traditional interfaces by changing configurations of invoking processes. Specialized actuation agents are available to implement a different type of action-taking. For example, in service intent management, a proposal may be expressed by a Topology and Orchestration Specification for Cloud Applications (TOSCA) model and the actuation would therefore be orchestration.

In all cases whenever multiple agents are implemented for a given closed loop stage, there should be some mediation layer to decide which agent should be executed, or the outcome of which agent should be considered if more than one agent is employed. This can be decided by the knowledge base and the decision about which agent to be considered can change over time according to the condition of the environment, the performance of the closed loop or any other policy specified.

6.4.2 Knowledge base-centric closed loops

The entire closed loop shown in figure 6.4.1-1 is based on agents creating and consuming knowledge. Agents both react to knowledge changes and create knowledge. For example, proposal generation is initiated by the occurrence of the type of issue that the proposal agent was implemented to handle. It delivers a proposal for the issue through the creation of a proposal description in the knowledge base.

The knowledge base becomes a central piece of the closed loop execution, and it is responsible for all data exchange between the closed loop stages. In figure 6.4.1-1, the arrows between Measurement, Issues, Solutions, Evaluation, Actuation, represent the logical flow of data; but in a knowledge base-centric closed loop all flows are driven by knowledge exchange between the closed loop stages and the knowledge base. This allows more flexible and dynamic execution of the closed loops. The main difference between knowledge base-centric closed loops and other closed loops is that the data and control flow between closed loop stage is driven by the knowledge base in a knowledge base-centric closed loop; in this case the knowledge base is more than a simple datastore, but also included reasoning capabilities to interpret different knowledge objects, which in most cases are coded as ontology graphs that leverage existing tools.

Intents are modelled in some standards as ontology graphs that represent knowledge about the service requirements [i.5]. With this modelling, a knowledge base that is based on ontology graphs can be leveraged to enable a complete knowledge-centric implementation of the closed loop. For that, ontologies can also be used to represent the network state, the issues, proposals, predictions, and all types of knowledge needed to create the flows between closed loop stages. This allows agent coordination through a generic logical-reasoning-based implementation.

This type of knowledge-centric closed loop is highly self-adaptive to changing intents. If respective agents are available, the machine-reasoning-based IME will autonomously utilize them by composing a custom, ad hoc workflow of agents. Moreover, the knowledge-centric closed loop in figure 6.4.1-1 is not specific to any particular management domain; it is, rather, a template for implementing any IME in any management domain. The creation of an IME for a particular MD is primarily an implementation task of domain-specific agents that need to be coded for each closed loop stage.

6.5 Hierarchical coordination between closed loops in resource constrained environments

The problem "resource locality and scarcity on closed loop automation" (see clause 5.5) notes the problem on having closed loops acting on resources which spread across the IoT-edge-cloud continuum. The managed resources located on the extreme-edge may provide scarce compute capabilities. These capabilities are clearly insufficient to instantiate and operate closed loops, considering the complex and resource-consuming analysis and decision-making activities that these closed loops are presumed to execute. However, relying only on a remote closed loop induces several problems: higher risks of disconnection, higher latency, and additional use of bandwidth.

To solve this issue, a hierarchical system can be established. Such system is represented in figure 6.5-1.

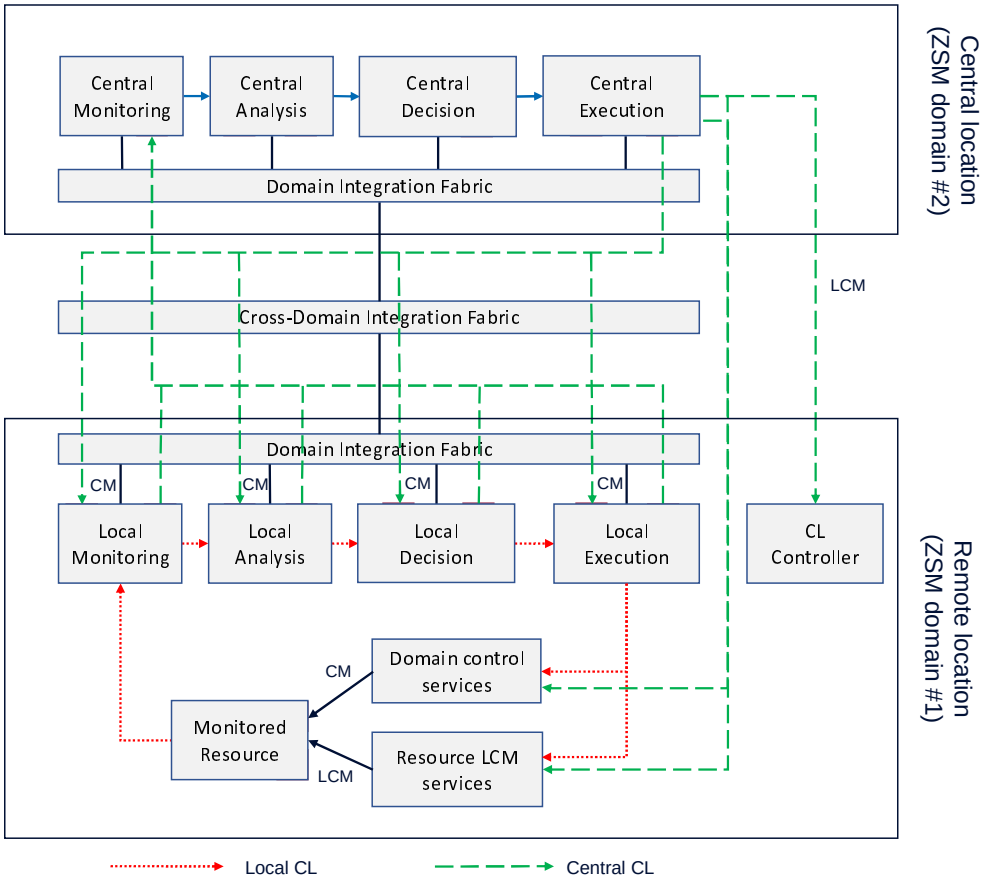

Figure 6.5-1: Hierarchical closed loop architecture

As seen, for the management of resources hosted in the remote location (e.g. extreme-edge node), it is proposed to set up a local closed loop. The scope of this closed loop is limited to fast yet simple analysis and decision-making activities. The objective of this local closed loop is to provide decent level of management in a timely manner, leveraging the limited computing capabilities provided by the resources available in this remote location. The local closed loop will also be able to work autonomously if/when a disconnection from the central closed loop occurs. As for execution, the closed loop may trigger corresponding management actions towards managed resources making use of ZSM domain control services (see clause 6.5.6 from ETSI GS ZSM 002 [i.9]). These actions include either configuration management actions (see clause 6.5.6.2.1 from ETSI GS ZSM 002 [i.9]), or resource lifecycle management services (see clause 6.5.6.2.2 from ETSI GS ZSM 002 [i.9]) or both.

In parallel, the data produced by this local closed loop can be monitored by a central closed loop. The nature of the data that the central closed loop chooses to monitor is open and may evolve depending on the needs and the amount of network bandwidth dedicated to this monitoring. For example, the central closed loop can decide to monitor only the data produced by the local closed loop's monitoring stage. Likewise, the central closed loop can also decide to monitor the data produced across all the stages (e.g. monitoring, analysis, decision, execution) of the location closed loop.

From a solution standpoint, different options are plausible:

- Option 1: Using one of the solutions already reported in clause 6.2.2, i.e. coordination of interdependent closed loops. The solutions can be used as-is, by simply mapping lower and higher closed loops to local and central closed loops, respectively.
- Option 2: Use a new solution tailored to the specificities of the extreme-edge. The solution description is as follows: Upon analysis of monitoring data, the central closed loop decides to act upon the managed resources available in the remote location. Here there exist two plausible flavors:
    - Flavor 2A (direct mode): The central closed loop interacts with the remote location directly, i.e. by consuming the ZSM domain control services exposed by this remote location. This option allows the central closed loop to access the remote location resources, and to execute configuration and/or lifecycle management actions on them.
    - Flavor 2B (indirect mode): The central closed loop interacts with the remote location indirectly, i.e. via local closed loop. In this interaction, the central closed loop approaches the local closed loop with either configuration management actions (e.g. changing the scope of monitoring collector, updating weights for the AI/ML models used in analysis and decision engines) or lifecycle management actions (e.g. deploying a new monitoring collector). Both types of actions can target any stage in the local closed loop, as noted in figure 6.5-1.

In some cases, the actions range of central and local closed loops can overlap, when they try to execute management actions on the same managed resources. This conflict situation had already been identified and is documented in clause 8.2.3 of ETSI GS ZSM 009-1 [i.2]. To solve this situation, the following mechanisms can be applied:

- In case of going for option 1, using the coordination mechanisms defined therein.
- In case of going for option 2, avoiding the central closed loop to directly act on managed resources (flavor 2A), delegating execution to the local closed loop (flavor 2B) instead.

7 Recommendations

7.1 Cognitive closed loops

Cognitive closed loops benefit from applying machine learning concepts in any of its stages making the closed loop operations more adaptive and self-learning. It is essential to address the extended requirements of cognitive closed loops and support services and capabilities required to configure and manage the cognitive closed loop.

Recommendation 1:

- Define cognitive closed loop and the services and capabilities that it needs.

Recommendation 2:

- Specify a mechanism to configure cognitive closed loop.

NOTE 1: Examples of capabilities that may be configured are:

1) data requirements for cognitive closed loops;
2) knowledge stored and/or extracted;
3) action-related aspects such as timing, impact, efficiency; and
4) use of multiple analytics components within the cognitive closed loops.

Recommendation 3:

- Specify a mechanism to manage cognitive closed loop.

NOTE 2: Examples to monitor the time required to achieve the expected results post action is triggered, reports on efficiency, impact, etc.

Recommendation 4:

- Specify escalation or fallback mechanism for cognitive closed loops to handle abnormal conditions.

NOTE 3: Examples of abnormal conditions may include insufficient accumulated knowledge.

7.2 Composition and coordination of interdependent closed loops

Coordination of interdependent closed loop is essential in achieving reduction in conflict scenarios arising from multiple closed loops which acts on same or related managed entities. Having external coordination is not sufficient since these mechanisms assumes the individual closed loops as independent and may not be able to address the interdependent scenario and due to complexity arising from scaling. As explained in clause 6.3, the challenges in coordination of interdependent closed loop can be achieved by grouping of these interdependent closed loops with each group handling the coordination within the group and external coordinator is involved in coordinating the actions from each of these groups.

Recommendation 1:

- ZSM Closed Loop Automation framework should support mechanisms to group interdependent closed loops by making use of the attributes defined under clause 6.2.1.

Recommendation 2:

- Specify common mechanisms to exchange information between the interdependent closed loops to address the requirements highlighted by the scenario under clause 6.2.2.

Recommendation 3:

- Specify mechanisms to coordinate the different stages of execution for different closed loops belonging to the same group.

Recommendation 4:

- Specify ways to convey instructions with the control flow sent by the authorized consumer and/or exchanged between the interdependent closed loops to address the requirements highlighted by the scenario under clause 6.2.4.

7.3 Dynamic composition of closed loops

Dynamic composition of closed loops allows ZSM framework consumers to request for made-to-order closed loop by combining different closed-loop components from different vendors.

Recommendation 1:

- Specify Closed Loop Controller (CLC) as the entity responsible for dynamic composition and instantiation of closed loops.

Recommendation 2:

- Specify mechanisms for a Closed Loop Controller (CLC) to expose means by which Closed-Loop Components are enabled to register capabilities, input/output supported by the closed loop components.

Recommendation 3:

- Specify mechanisms for a Closed Loop Controller (CLC) to allow ZSM framework consumer to request for "made-to-order" closed loops (M2O-CL) based on the expected inputs and information on the required closed loop components and stages.

Recommendation 4:

- Specify mechanisms for a Closed Loop Controller (CLC) to validate the request from ZSM framework consumer and provide feedback on the validity of the request.

Recommendation 5:

- Specify mechanisms for a Closed Loop Controller (CLC) to allow ZSM framework consumer to query information about made to order closed loop capabilities that are supported by the Closed Loop Controller.

7.4 Intent-based closed loops

Closed loops and intents are two critical automation enablers that should work together to improve the autonomy of service and network management. Some steps still need to be taken to better specify intent-based closed loops within the ZSM frameworks, as follows.

Recommendation 1:

- Intents should be considered as a source for the definition and specification of the (composition of) ZSM closed loop goal(s).

Recommendation 2:

- The ZSM closed loop enablers defined in ETSI GS ZSM 009-1 [i.2] should be leveraged for the realization of intent management entities (according to definition in ETSI GR ZSM 011 [i.4]) within the ZSM framework.

Recommendation 3:

- Identification of gaps for the combination of intents and closed loops on the existing ZSM 009-1 closed loop enablers (CL governance and CL coordination) should be done and solutions should be developed for solving those gaps.

Recommendation 4:

- The use of semantic knowledge representation (e.g. ontology graphs) should be considered within the ZSM closed loops to facilitate the exchange of data and control between ZSM closed loop stages.

Recommendation 5:

- Specify ways to facilitate the creation of intent-based closed loops, e.g. semantic web technologies such as RDF and OWL.

Annex A:
Change History

Date	Version	Information about changes
September 2019	0.0.1	Early draft: agreement on the skeleton
October 2020	0.1.0	Insertion of contribution ZSM(20)000154r1_ZSM009-3_Advanced_CL_features Removed Editor's note: Update the figure to reflect agreement on text changes for data sources
July 2022	0.2.0 0.2.1	Update to clause 5: new sub-clauses Insertion of contribution ZSM(21)000364_ ZSM009-3_Composition and Coordination of Interdependent Closed Loops
September 2022	0.3.0	Insertion of contributions - ZSM(22)000250r3 - ZSM009-3 Sec 5.4 Dynamic Composition of Closed loops - ZSM(22)000277r2 - ZSM009-3 Sec 6.4 Dynamic Composition of Closed loops - ZSM(22)000284r2 ZSM009-3 - Intent-based closed loops Update to Informative references and acronyms Harmonization of figure and table references Editorial and style corrections
September 2022	0.3.1	Fixed omission of figure 5.4-1 of ZSM(22)000284r2 ZSM009-3 - Intent-based closed loops when updating to latest draft
June 2023	0.4.0	Integration of contributions - ZSM(22)000370r2_ZSM009-3_Sec_7_2_Composition_and_coordination_of_interdepend - ZSM(22)000371r1_ZSM009-3_Sec_7_3_Dynamic_composition_of_closed_loops - ZSM(22)000375_ZSM009-3_Sec_6_3_3_replacing_must - ZSM(22)000381r3_ZSM009-3_Sec_4_Next-generation_closed-loop_operations - ZSM(22)000382r1_ZSM009-3_Editorial_changes - ZSM(22)000383r1_ZSM009-3_Sec_7_1_Cognitive_closed_loops - ZSM(22)000392r2_ZSM009-3_Section_5_Resource_locality_and_scarcity_on_closed_ - ZSM(22)000393r3_ZSM009-3_Section_6_Add_potential_solution_for_handling_resou - ZSM(22)000398r2_ZSM009-3_Section_7_4_Recommendations_for_intent-based_closed - ZSM(23)000106_ZSM009-3_Changing_clause_5_2_coordination_of_CLs - ZSM(23)000107_ZSM009-3_Modifying_6_3_1_grouping_interdependent_CLs Update to Informative references and acronyms Harmonization of figure and table references Editorial and style corrections
July 2023	0.4.1	Integration of contribution: - ZSM(23)000108r4_ZSM009-3_Modifying_6_3_3_Example_workflows Change work item status to Final draft
July 2023	0.4.2	New draft due to mistake in attachment of v0.4.1 as final draft v0.4.2 is same as v0.4.1 from content viewpoint

History

Document history		
V1.1.1	August 2023	Publication