I^2BN: Intelligent Intent Based Networks

Péter Szilágyi

Nokia Bell Labs, Budapest, Hungary
E-mail: peter.1.szilagyi@nokia-bell-labs.com

Received 22 January 2021; Accepted 09 March 2021;
Publication 31 May 2021

Abstract

Intent based network management reduces the complexity of network programming from a growing set of deeply technical APIs to context-free high-level objectives that the network should autonomously achieve and keep. The practical implementation of an intent based network requires substantial automation technology embedded in the network. Automation should cover the entire lifecycle of intents, from their ingestion to fulfillment and assurance. This article investigates the feasibility of automatically assembling interworking implementation units into intent specific automation pipelines, where units are reusable self-learning closed loop micro-services with self-declared capabilities. Each closed loop may gain knowledge and respond to dynamically changing network conditions, thereby enabling network autonomy in reaching the declared intent objectives. The human-network intent interface for expressing intents is proposed to be based on the aggregation of the deployed network and service automation capabilities, rather than a formalism decoupled from the actual network implementation. This principle removes the ambiguity and compatibility gap between human intent definition and machine intent fulfillment, while retaining the flexibility and extendibility of the intents offered by any specific system via onboarding additional micro-services with novel capabilities. The concepts discussed by the article fit into the architecture and closed loop work items already defined by ETSI ZSM and provides considerations towards new areas such as intent driven autonomous networks and enablers for automation.

Journal of ICT Standardization, Vol. 9_2, 159–200. *River Publishers*
doi: 10.13052/jicts2245-800X.926

Keywords: Intent, I^{2}BN, network and service automation, network intelligence, ZSM.

1 Introduction

Intent based service and network management is a paradigm shift in the telecommunications industry. On the surface, intent refers to the way operators and enterprises interact with network technology to express their expectations about services, efficiency and system behaviour in general or related to particular use cases. As opposed to today's programmable networks, which are governed through condition-action policies and manual mastering of ever growing set of Application Programming Interfaces (API) and exposed capabilities, intent promises to enable network users declare their end goals with the network, rather than providing a list of instructions to execute without being able to articulate why those instructions were provided. The network itself should figure out when and what actions or reconfigurations to execute, in which technology or vendor domains, for which end users, terminal devices or geo-area. Therefore, below the surface of an intent based network, there is automation and, increasingly, autonomy. While automation means more the recitation of previously pre-programmed workflows, albeit enriched with conditions and some degree of adaptation capability via scripting, autonomy refers to a whole new level of network responsibility. It is about delegating massively complex inter-domain, multi-vendor and multi-technology orchestration, service management, real time parameter configuration and resource management to the network, in the face of a dynamic flow of demands and users to be served. Essentially, as we argue in this article, the full rollout of the intent paradigm should eventually transform programmable networks to self-programming networks.

The inception of intent based networking can be traced back to the solid foundation of policy-based networking [1], evolving through intent based policy management [2] to intent based networking [3]. The terms intent based networking and intent based orchestration have been in use by multiple standard organizations, including Internet Research Task Force (IRTF) [3, 4], 3rd Generation Partnership Project (3GPP) [5, 6] and in the realm of Software Defined Networking (SDN) such as Open Networking Foundation (ONF) [7] and Open Network Operating System (ONOS) [8]. The definitions of intent in each work group are slightly different and specific to their respective core technology. Nevertheless, intent is commonly understood to be a high-level declaration of business or service level objectives to be achieved by the network or desired behaviour the network is expected to be conform with. The intent does not carry any concrete actions or configuration steps that the network should take in order to achieve the described goals. It is also recognized that in order for intents to be a feasible communication abstraction between humans and networks, networks are required to implement certain mechanisms of automation and intelligence, potentially by using artificial intelligence and machine learning techniques [9]. However, none of the above sources provide a deep technological view on what method and apparatus would actually make a network capable of interpreting the intents in the first place, and to dynamically derive the necessary actions or configuration and apply them to existing software and infrastructure. Self-learning enablers that are key for autonomous network side decision making and action monitoring are also kept at very high level.

This article aims to progress the discussion around intents by publishing concrete ideas and proposals. We inspect what are the key ingredients that make intent work so well between humans and derive key requirements from the observations in the context of a brief historical overview of intent's evolution throughout the telecommunications history. We also analyse what is the challenge in managing todays networks that are increasingly software defined and programmable yet are far from being intent based. The main contribution of the article is a hierarchical intent management architecture built on the foundations of self-learning closed loops [10] with automation extended to the assembly and orchestration of the closed loops themselves. A key principle introduced by the article is to reverse the usual direction of looking at how human defined intents are supposed to be unambiguously interpreted by machines. Instead of aiming for a "generic" network intelligence that can understand intents in any form and unlimited richness of expression, we argue that it is more realistic to provide means for synthetizing an intent interface that reflects the aggregated capabilities of the what the network could do autonomously. This interface would evolve as the network management services are extended by onboarding new developments of micro-services with automation capabilities.

The rest of this article is organized as follows. Section 2 starts with a brief historical overview of intents in telecommunications and coins the term Intelligent Intent Based Networking (I^{2}BN) by arguing that intents drive the need for more network side intelligence. It is followed by discussing what are the challenges of managing a programmable network and what is the benefit of intents. Next, system requirements are derived from analysing the human-network intent interface, leading to the definition of an intent based management architecture, the life-cycle of intents and the interactions of intent management with other network operations. Section 3 dives in implementation aspects, starting by laying out a set of practical design principles for intent based networks and providing corresponding high-level solutions. Section 4 is a technology deep dive into how the self-learning closed loops may be built from reusable micro-services, and what are the consequences of using machine learning for realizing parts of the functionalities. Finally, Section 5 concludes the article.

2 Towards I^{2}BN

This section argues for the need of substantial intelligence and automation within the network for intents to become a value-added construct and lays out a vision of what is referred to as Intelligent Intent Based Network or I^{2}BN.

2.1 What is Intent?

In real life, intent is an objective given by someone to another person, capturing an objective or state to be reached without providing the means to achieve it. The other person is supposed to autonomously derive the necessary tools to be utilized and the necessary steps that need to be undertaken. Feedback about the progress and end result as well as major obstacles are expected to be delivered in the opposite direction. In human life, professional or private alike, intent is the essence of collaboration and delegation.

Intent works for humans due to a number of reasons. First, humans are intelligent, both in formulating the intent and comprehending it. Yet this intelligence would not be enough without shared context and knowledge between those participating in the exchange of the intent. That is to say that the interpretation of an objective may depend on the environment and shared history between the persons. Additionally, humans tolerate incomplete information sets, which makes them able to start the execution of a task even if not every detail of the execution path is laid out upfront or there remain open questions. Finally, dialogue is an invaluable capability that we use to resolve ambiguity as we proceed with intent execution, asking for clarification from the intent's provider, or reporting on an unforeseen circumstance that may require the reconsideration of the intent. Recognizing that in a given context we do not have the enough information or the right skills to decide or act is another capability people naturally have.

Intent based networking is a paradigm that aims to reformulate the management of network domains, e2e networks or services, along the principles of how the human intent works. Today's networks are programmable, yet they need programmers. Networks allow the definition of policies, which are essentially conditional actions to be executed at pre-defined scenarios. Engineers and operational staff need to define when and what the network functions should do in meticulous details, at multiple resource and technology layers and on massive scales. No entity in the network, including the management layer, has a full representation of the overall goals, towards which the individual policies represent particular and incomplete programs. In contrast, intent based networks are to allow the definition of much higher level and, therefore, more persistent, sometimes invariant and abstract objectives, without requiring instructions on how to reach them. This of course requires additional capabilities within the intent based network itself, namely a level of intelligence, which, speaking of machines, will be artificial. But, as with humans, no intelligence itself is sufficient to carry out an intent without the ability to comprehend the context, knowing the potential actions available to manipulate the scenario, and being aware of the expected outcome of each action in the given context. In networks, context awareness means sufficiently real time and detailed measurement and monitoring on resource, network function, technology domain, e2e and service level. Knowing the potential actions means to have the necessary APIs for resource management, domain controllers, Quality of Service (QoS) provisioning and changing, slicing and more. Knowing the outcome of the potential actions means the ability to calculate, evaluate and rank which action at which scope would be most efficient and impactful in aligning the network and services with the goals formulated by or derived from the intent. An additional capability, which is the substitute for human intelligence and language, namely the comprehension of intent and its mapping to actions that make sense in any moment, is needed for bridging the major abstraction and automation gap between intent and APIs. This is a novel requirement compared to non-intent based networks (such as those having only policy or workflow automation).

Intent based networking itself is not a new term. Yet existing proposals have usually a narrow domain specific use case driven automation scope, leaving the above identified abstraction and automation gap open in general, especially when considering e2e service level intents that are supposed to govern network behavior across multiple technology and vendor domains. Therefore, to highlight the difference, this article proposes the term Intelligent Intent Based Network or I^2BN as reference to the above vision that brings intent fluidity in networks on par with how naturally humans interact based on intents.

2.2 A Brief History of Intent

During the last decades, the social acceptance and utility of telecommunications in private and business life have evolved alongside the technology itself, as interaction with the technology has become increasingly streamlined. The technology went through many stages (Figure 1), from its inception in 1876 by A. G. Bell as an analogue point-to-point channel and telephone exchanges driven by manual switchboards; through 2G, the first digital system for mobile human-to-human voice communication; 4G, with smartphones and mobile Internet; and now 5G with massive Internet of Things (IoT), Ultra Reliable Low Latency Communications (URLLC) and machine to machine communication services. Each of these major steps required adaptation in how end users interacted with the technology, where the means to interact essentially constitute the intent interface from a consumer's perspective. In the early landline days, one could only call places and ask for a person to come to the phone. Intents were then basically entries in the phone book that could be dialed (or, even earlier, asked from the switchboard operator). Later, with 2G and the personalization of devices came the intent to call or text individuals, which essentially kept the customer services at the same complexity level. Then, with 4G, smartphone and touchscreen evolution came a seemingly explosive evolution of digital content and interactions brought to the mobile platform, yet considering the underlying telecommunication services, most of it was and continues to be running on best effort packet switched data connectivity. Likewise, operators of the telecommunications systems had to adapt their interactions and means of achieving their business targets, mostly sculpted by the evolving means of automation [9]. From literally having humans in the loop to do the cable switching, they went through manually triggered batch scripts to big data processing and analytics, which support a solid level of workflow automation in the policy- or rule-based systems. Yet, current network management solutions require the operator to specify the lifecycle of any particular automation loop that is to be executed, without such loops forming autonomously by the system based on high-level objectives. 5G integrates various machine type communications and means of mixing services on the same infrastructure, a leapfrog from 4G's one network – one service principle. The more diverse the services though, the higher the risk of fragmented user and management interfaces (both human-

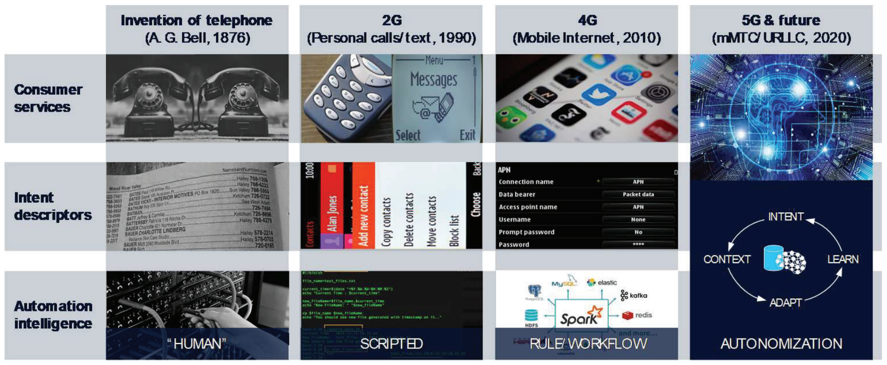
Invention of telephone (A. G. Bell, 1876)
2G (Personal calls/text, 1990)
4G (Mobile Internet, 2010)
5G & future (mMTC/URLLC, 2020)
Consumer services
Intent descriptors
Automation intelligence
INTENT
CONTEXT
LEARN
ADAPT
"HUMAN"
SCRIPTED
RULE/WORKFLOW
AUTONOMIZATION

Figure 1 The brief history of intent and automation in telecommunications.

or API-centric). Thus, for the first time in using or managing networks, intent and intent based automation may not only be a convenience layer but a rather instrumental and deeply integrated artifact to prevent management complexity from limiting the accessible and exploitable capabilities of the system in terms of efficiently serving versatile demands. This is even more pressing as with 5G cellular technology gets integrated into non-telco businesses such as enterprises and verticals, where the primary concern and knowledge is not rooted in telecommunications as opposed to a classical Communication Service Provider (CSP) with telco expertise. Therefore, high-level interfaces and intelligent intent based interaction patterns backed by a much more autonomous system than today are key to unlock the technological and business potential of telco and vertical fusion.

2.3 Challenges of Managing a Programmable Network

It was mentioned in Section 2.1 that today's networks are programmable. That is, network functions, U-plane and C-plane, as well as management services expose standard or proprietary N-bound interfaces that can be leveraged to cause the network apparatus to execute specific pre-defined programs. A good example is SDN, where the programmable apparatus consists of SDN controllers exposing an N-bound API that can be used to query the controller about network status or provide configuration in a form that is more abstract than the APIs of the particular transport network equipment under the SDN controller.

One point to be made is that programmability is a necessary enabler to build more autonomous systems. Another point, however, is that raw

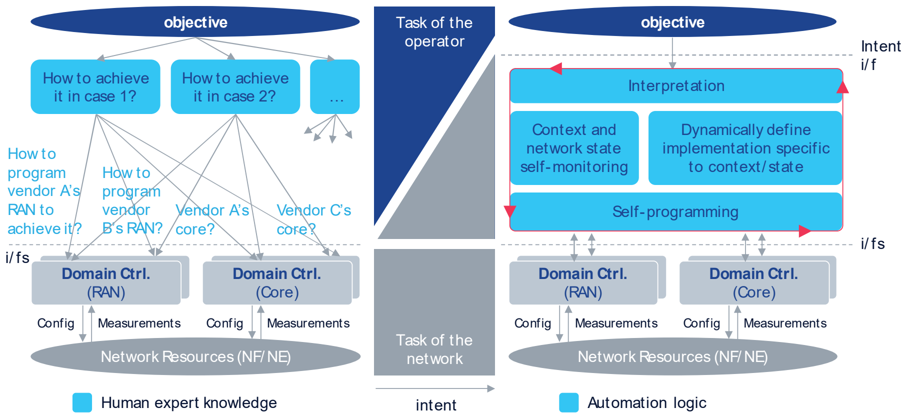
objective
How to achieve it in case 1?
How to achieve it in case 2?
...
How to program vendor A's RAN to achieve it?
How to program vendor B's RAN?
Vendor A's core?
Vendor C's core?
i/fs
Domain Ctrl. (RAN)
Domain Ctrl. (Core)
Config
Measurements
Network Resources (NF/NE)
Human expert knowledge
Task of the operator
Task of the network
intent
objective
Intent i/f
Interpretation
Context and network state self-monitoring
Dynamically define implementation specific to context/state
Self-programming
i/fs
Automation logic

Figure 2 Challenges of programmable networks and the benefits of the intent paradigm.

programmability is not sufficient to achieve high level of automation, that is, where automation goals are beyond configuration parameters that can be committed to the registers of traffic-facing equipment in an atomic step.

The challenge of managing a programmable network and what is the benefit of the intent based paradigm are presented in Figure 2. Whenever the operator of the network has a high-level technical or business objective (e.g., providing good Vehicle-to-Everything (V2X) service in an area [5, 6], or achieving efficient resource utilization without compromising customer experience unless it would cost excessive amount of resources), that implementation of the objective has to be broken down to specific cases. With policy-based management (the state of the art of network programmability), each case has to represent a condition and action, both of which are need to be expressible by means of the collective set of APIs exposed by the network functions to be involved in achieving the objective. That is, the operator has to plan out the execution steps that would (presumably, and hopefully) reach the objective, in details. Also, as such planning involves many unknowns and simplifications, it may not be entirely successful, which would require (semi-)manually redoing the entire loop. That is, we are currently dealing with open-loop network programming at least in the management layer. An additional dimension of complexity is the cardinality of the rules that need to be programmed to articulate even a single high-level objective, not to mention the many (and potentially conflicting) objectives a business has to balance. Consequently, such network programming exercises are not even near real-time and are not repeated frequently, leading to the following "best" practices:

- An ever-growing collection of use case specific independent APIs and their closed loop implementation blueprints by vendors
- Long development cycles, narrow scope, hard-wired management or optimization loops (such as with Self-Organized Networks)
- Complexity exposed through increasing number of parameters with no universally optimal value set
- In most of the cases automation means process automation

In contrast, the intent paradigm draws an interface at the level of objective formulation and claims that the programming step(s) should be encapsulated by the network itself. In order to allow incremental development and deployment of such a future system, reusing the existing APIs (which are currently exposed to human experts) is recommended. This paradigm shift by I^2BN implies no pre-defined relationship between API providers and consumers, as any concrete action has to be a function of the future demand, traffic mix to be served, configurations, resource availability and load, realized service qualities, etc. As these depend on external factors such as user mobility, application and server behaviour, content to be transferred, etc., they cannot be prescribed in advance; they have to be derived on the spot as they happen. Therefore, it cannot even be described which APIs will need to be called to collect the necessary insight and which APIs will need to be used to actuate any changes to maintain the objective (that's exactly the problem of policy-based systems). This further implies that the network has to become autonomous and self-programming as depicted in Figure 2, becoming capable of deriving these insights and actions through measurement/data collection and configuration setting APIs. The technical enablers for such system include at least the followings:

- Means for efficient data collection and correlation. This implies machine readable syntactical and semantical annotation of data as well as means to identify data collected from various sources that share a common state and context by serving the same end-to-end data flows.
- Ability to consume and process analytics that is context/intent aware at near real-time speeds at scale [3]. This is required to support micro-service architectures and programmatic pipeline orchestration and deployment necessary to compose purpose-build insight driven closed loops.

- Software based architecture and programmability (softwarization, SDN, etc.), as mentioned before.

In addition to the above functional considerations, there are novel trust and security implications posed by an autonomous system, including but not limited to:

- Autonomy and trust (network taking business critical decisions); ownership of responsibility.
- Security with Machine Learning (ML) models (trained on untrusted data, deployed in micro-services across multiple clouds, integrating open source).

2.4 Requirements Driven by an Intent Based Human – Network Interface

After reviewing the management challenges of today's programmable networks, let us turn to the requirements towards an intent based human-network interface. As this interface is the visible part of an intent based system, it is logical to start deriving requirements from the expected interactions. Nevertheless, requirements to the system beyond the interface are also collected. We start by listing motivations and expectations of potential users of such a system, focusing on business users, operators and enterprises, as they are the primary consumers of management services.

Motivation: Different roles in a CSP such as planning, operations support system (OSS) and business support system (BSS), sales, marketing, wish to continuously evolve the services to provide customer value and business revenue.

➔ Requirement: Support all CSP roles to define their objectives using a language/API that is close to their own domain abstractions and support their conventional decomposition of systems and services.

Motivation: Verticals and Enterprises (industrial networks, private networks) with to increase revenue through adding value to own business and increase production efficiency in their own technology.

➔ Requirements: Clean-cut high-level interfaces between industry users and telco technology to support telco integration without deep telco knowledge.

Motivation: Bi-directional interfaces, human to machine (H2M) and machine to human (M2H), are needed to enable not only the definition of business, service or resource level objectives, but also to receive insight on system state, intent fulfilment and assurance.

➔ Requirements: Provide meaningful feedback actionable for the entity who provided the original intent (e.g., to enable reconsidering it) rather than logging deeply technical error messages.

Motivation: Simplicity, safety & trust to delegate management execution (fold the complexity into the system) is inevitable if the user facing interface is intent based.

➔ Requirements: Automatically reach the objectives based on the context/state of users, devices, resources and potential actions. Harmonize actions to reach multiple objectives (related to multiple intents), resolve or indicate conflicts.

Motivation: Therefore, mechanisms of trust, supervision, direct control and manual override are expected to be exposed and integrated by an intent based system.

➔ Requirements: Enable to directly interact with specific resources and override auto-derived contextualized targets in case of unforeseen circumstances (e.g., sudden business need with contracted requirements that cannot be derived automatically).

Based on the above requirements, we proceed to discuss design and architecture principles of a system that could fulfil them.

2.5 Architecture

A potential architecture for bridging the gap between high-level objectives and operating the APIs of the existing domain and resource controllers is depicted in Figure 3.

The main architectural components of the system are a set of Intent Managers, which are organized into a hierarchy. Intents related to end-to-end (e2e) services or expected e2e network behaviour should be received by an entity that is entitled to act in e2e. Therefore, at the top, the customer (i.e., user) facing Intent Manager is referred to as the E2E Intent Manager (IM). The E2E IM is on top of a next layer or more specific Intent Managers (having the scope of a domain, technology, administrative area, etc.). This second level of IMs is necessary as no single entity is expected to have the full detailed view of the entire system, including all vendors, technology domains and resources. E2e intents received by the E2E IM have to be analysed,

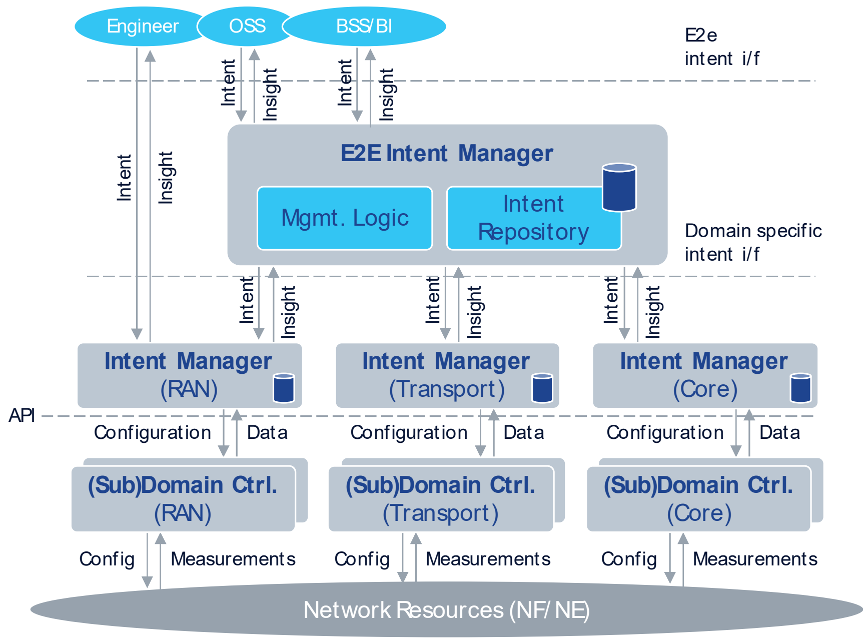

Figure 3 Intent based management architecture.

segmented and cascaded down towards those more specific IMs. While the E2E IM remains responsible to the fulfilment of the e2e intent, it has the freedom to decompose and delegate (domain) specific parts of it to specific IMs. In this way, E2E IM uses the API of the next level of IMs. In turn, the specific IMs use the APIs existing domain controllers, such as Radio Access Network (RAN), Transport or Core domain controllers (e.g., orchestrators, SDN controllers) to make the necessary calls that would fulfil the part of the intent that was delegated to them by the E2E IM. Just as intent is decomposed and travels down, measurements and insights are collected and streamed upwards in the hierarchy of IMs. This is necessary to prevent low level details within a particular domain or resource context being exposed to the operator. Thus, the E2E IM has the task to synthetize and present service, network and resource state related to an intent with abstraction level and terms matching that of the original intent.

So far only e2e intents were considered. However, an intent based system should also support receiving more specific intents or even direct configuration on domain controllers or resource level. Such requests are expected to originate from domain experts who know what they are doing and therefore can also consider the consequences of interacting with the system on lower levels. Still, allowing such interactions is essential to retain access to the system's full configuration space and enable quick overrides, both in cases of handling temporary or exceptional demands with direct and known configuration needs, and to override any decision taken by the system for any reason.

In alignment with the Zero Touch Network and Service Management (ZSM) architecture [10], the E2E IM may be part of the E2E Service Management Domain, whereas the next layer of IMs could be part of specific Management Domains. Communication between the IMs is possible via the ZSM Cross-Domain Integration Fabric.

2.6 Intent Life-cycle and Interaction with Network Operations

The lifecycle of an intent is discussed in the wider context of planning and operating network services, as depicted in Figure 4. The network itself can be regarded as a set of hardware and software assets (including not only U/C-plane functions but also management services including closed loops), which interplay with their current configuration and the current demand (user traffic) to serve. In pre-launch, usually planning activities are carried out to design and dimension the key metrics of the network. Basic service blueprints are also provided, along with the first bootstrapping configuration. In an intent based system, basic intents may already be provided. The network transitions into and operating network as it starts to serve user demand. As the user demand is usually dynamic in nature, the network needs to adapt its operation to provide the required service. There may be various adaptation loops. Traditionally, the operator may provide configuration changes based on high level Key Performance Indicators (KPI), faults, anomalies or other performance degradations [11, 12]. As this is an open loop process, it is slow (many days or weeks per cycle). Additionally, and more interestingly, various closed loops (i.e., software-based automation) in the system may take proactive or reactive actions to keep the system within proper operation bounds. With intents, such loops are expected to be driven by the objectives described in the intents and the current state of the network, i.e., the level of fulfilment regarding of said objectives. Those closed loops are necessarily part of the intent managers as it requires that the intent (which is more static and context-free) is interpreted in the current context of network resources, e2e and domain level state, and actions that make sense in the given context are triggered (usually via API calls to the domain managers).

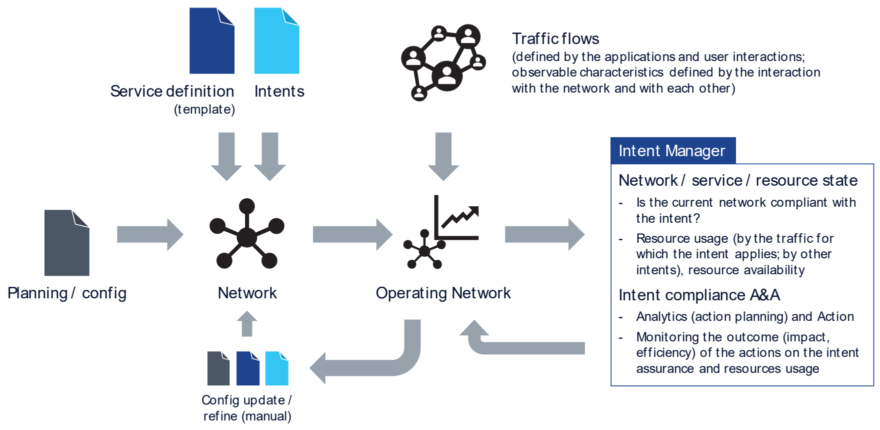
Service definition
(template)
Intents
Traffic flows
(defined by the applications and user interactions; observable characteristics defined by the interaction with the network and with each other)
Intent Manager
Network / service / resource state
- Is the current network compliant with the intent?
- Resource usage (by the traffic for which the intent applies; by other intents), resource availability
Intent compliance A&A
- Analytics (action planning) and Action
- Monitoring the outcome (impact, efficiency) of the actions on the intent assurance and resources usage
Planning / config
Network
Operating Network
Config update / refine (manual)

Figure 4 Intent life-cycle and interaction with network operations.

Actions of an intent based system related to the lifecycle of intents may be categorized as intent fulfilment and intent assurance [3]. Intent fulfilment refers to the steps taken by an Intent Manager in context of receiving a new intent or an update to an existing intent (e.g., the E2E IM receives an intent from the operator, or a lower level IM receives an intent from the E2E IM). The goal of intent fulfilment is to bring the system's state in alignment with the new or updated intent's objective. The first action related to handing a new or updated intent is intent ingestion, which refers to receiving the intent via the exposed intent interface. It triggers actions such as validating the intent's syntax and semantics (API compatibility) and storing the intent in the IM's Intent Repository. Next, the intent needs to be translated, which means to analyse which part of the intent needs to be delegated by using services of other entities (lower ranking IMs, domain/resource controllers, orchestrators) and what actions may be taken directly by IM itself that originally received the intent (Figure 5). Delegation requires analysis based on the intent's goals and the APIs exposed by the potential other services, as well as based on the current context (network state, load, resource availability, etc.) of the system.

Note that Intent based automation does not replace existing technology/ domain-specific or protocol layer automations (e.g., Transmission Control Protocol (TCP) congestion control, fast reroute on transport link/node failures, dynamic routing, path protection/restoration) or self-configuration mechanisms (e.g., SON Automatic Neighbor Relation (ANR) [13]). Such mechanisms continue to exist and are expected to operate within their design boundaries. Instead, the IM interacts with other orchestrators only to use their

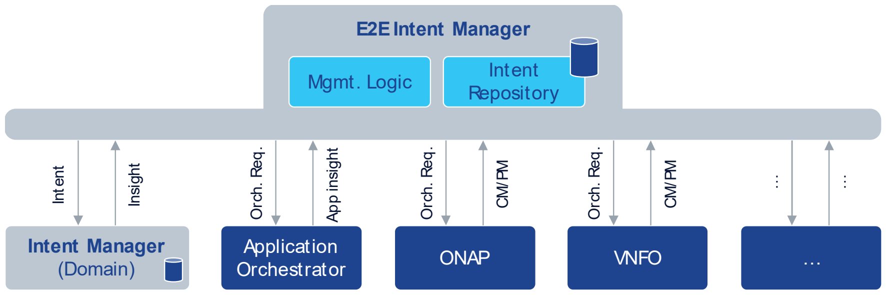

Figure 5 Relation of Intent Manager to other M-plane actors and management services.

services for intent fulfillment and assurance, not to manage those entities (Figure 5).

The lifecycle of some intents may end at the fulfilment case, if the intent's goal simply describes the availability or presence of a service. In that case, onboarding all necessary software artifacts (e.g., software images) and deploying the related (micro-)services (starting and scaling containers) completes the goal of the intent by definition. However, in other cases, the intent's objective attributes certain operational criteria (related to availability, quality of service, end user experience, etc.) in addition to the need of existence of a service. Those intents have their lifecycle tied to the operation of the referred service and may require frequent recurring actions executed via the APIs of the available domain/resource controllers to keep those operational criteria achieved. As such actions need to be triggered automatically by the Intent Manager who is responsible for that (part of the) intent, Intent Managers will need to leverage closed loops [14] (Figure 6). The closed loops may be created purposefully by the IM or may exist anyways but are utilized by potentially multiple IMs. The closed loops have to address network and service state monitoring, analytics and actions. Monitoring may include collection of performance management (PM) or fault management (FM) counters; resource, network and service state modelling; application insight; collected across e2e, domain and individual resource scopes. Collected data is then analysed. The purpose of the analytics is to assess the system's intent compliancy, that is, how much the intent's objectives are met, and to derive potential actions that would bring the system closer (ideally, to meet) those objectives. The technical enablers of such analytics will be discussed deeper in Section 4 in relation to self-learning closed loops [15].

IMs have to implement reporting towards the entity which ingested the intent they are working on. Reporting requires abstraction and aggregation

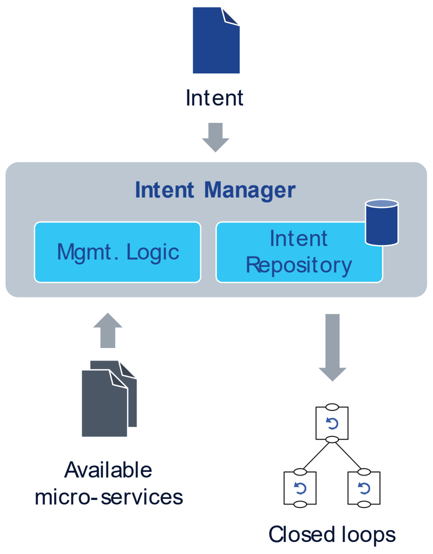

Figure 6 Intent Managers leverage closed loops for intent assurance and fulfilment.

of information to align the reported data with the ontology of the intent so that the report's semantics are comprehensible by the intent's author. This is essentially the inverse of intent translation but applied to measurements and insights related to the intent, not to the intent itself. Within such reporting capabilities, advanced Intent Managers may have an extended use of measurements. Besides evaluating the system's intent compliancy, measurements can be used to profile system capabilities. Such profiles may include, e.g., intrinsic limits on U-plane performance, resource capacities, amount of demand to be served within a given quality, etc. Profiles like that may be used to do a pre-launch assessment of a new or updated intent as part of the intent ingestion process. This assessment evaluates whether the objectives associated with a given intent are realistic and achievable by the system. As a trivial but intuitive example, if the intrinsic delay of a system within a given path is 20 ms but an intent with a 10 ms target is ingested, the receiving IM may immediately give a feedback to the intent's author that this intent is not realistic (at least at areas serviced by the given path). The feedback may indicate clues of what would be realistic, or, if the reason for likely undeliverable intent is the presence of high load or other high priority services taking up resources, indicate where is the conflict so that the intent's author may issue changes to other intents or arbitrate priorities.

3 Implementation Considerations

The relation of intents and closed loops has been hinted already as we discussed the lifecycle of intents and how they relate to network operations. Now this relation is explored more deeply and systematically with the aim of deriving more concrete design, architecture and implementation guidelines. We start from a list of numbered Design Principles (DP), each of which captures a key aspect of how an intent based system (i.e., the intent related part of it) could be realized in practice.

3.1 Design Principles

DP#1: Intents are not generic wishes expressed in isolation from any particular system, but they are based on declared and aggregated network & service automation capabilities (of a specific system). Such capabilities should be exposed to the operator in formal language to promote machine readability and eliminate ambiguity that is inherently part of natural languages. This design principle acknowledges two important differences between human and machine capabilities. First, human-like generic intelligence is not available in machines, therefore the freedom of thought and associations that can drive a human-to-human intent expression is not realistic on machine intent interfaces. Second, systems cannot evolve themselves or acquire new capabilities (as opposed to humans, who can autonomously acquire knowledge to fill the gaps in their capabilities); therefore, what any deployed system (set of services, available software images, hardware (HW) resources, etc.) can achieve is limited by its implementation and will only change through software development effort. Therefore, it is beneficial if the user of an intent based system is guided to author intents that already reflect what the system is able to achieve, to avoid user dissatisfaction upon freely entering a series intents that turn out to be incomprehensible or unimplementable (even though the intents may be perfectly valid when directed to a human engineer, it's just that the underlying system lacks implementation for them). It also helps the user discover the right syntactical and semantical terms when expressing intents, rather than the system trying to bridge the gap between under-specified or overloaded human terms and rigid fully specified machine APIs.

DP#2: Autonomy is increasing, supervision becomes coarser and trust in delegation is inevitable with intent based systems. That is, intents trigger automated decisions and actions in contexts/situations not explicitly prescribed in advance (as opposed to rules or policies). While this is the very reason behind the inception of intent based systems, the processes, workflows and best practices around managing such systems must also be changed and adjusted.

DP#3: Intents are implemented by means of closed-loop automation micro-services, not as a single monolithic cross-domain full-stack entity. This is in-line with contemporary software-as-a-service architecture guidelines, supports integration with cloud platforms and eliminates vendor lock-in. However, it also generates technical requirements on how the system can be composed from its parts, especially considering the high level of automation such composition needs to achieve. Some of these requirements are the following (more technical analysis will be given by Section 4):

- Micro-services expose services that define what type of automation they claim to be able to perform, and what type of inputs (intents) they can interpret. Such capabilities are better explicitly defined by the implementation of the micro-services and not puzzled together by a generic AI.
- One micro-service may use the services exposed by other micro-services (e.g., to collect measurements and other real time network state/context information; or to execute actions) to complete their operation. This is analogous to how common software functionality is collected in and published as reusable libraries or packages, which may themselves also use further libraries, etc.
- Each micro-service may internally implement self-learning (adaptation to context learned via self-monitoring) [15].

DP#4: Each intent is implemented by a recursively auto-generated closed-loop (CL) hierarchy, from top-level (where the intent is received, e.g., at e2e service level) to domain controller level.

- The hierarchy results in an upside-down tree with a single top-level micro-service (the one that receives the operator's input) per intent. Additional micro-services are recursively identified and added by each micro-service (starting by the top-level one) until domain level controllers are reached.
- The operator may interact with the CLs at any level (top-level is where the operator provides the input, but it may be already close to or at domain level).
- It is the responsibility of an Intent Manager to orchestrate closed loops together into a working pipeline along a given intent.

DP#5: Approach to conflict resolution during the lifecycle of a single intent and while handling multiple intents:

- Each CL receives higher level instructions related to a given intent from only one upper level CL (but each CL may provide derived intents to multiple lower-level CLs. Therefore, there are no inter-CL conflicts within the implementation of one intent.
- The same CL may receive multiple intents or instructions from upper level CLs (at top-level: the same micro-service may be the entry point for multiple intents). Each micro-service is responsible to resolve conflicts among its inputs. Therefore, all potential conflicts are intra-CL and should be handled by the CL's implementation. Implementations may compete in their ability to harmonize and fulfil more versatile set of intents and requiring less of the system's resources (compute, bandwidth, etc.) in doing that.

3.2 Implementation Solutions

After outlining five Design Principles, let's discuss potential implementation-oriented solutions and practical considerations that are addressing one or more of the DPs.

3.2.1 Auto-generating the CL hierarchy (DP#2,3,5)

Whenever a solution is composed of multiple interworking but potentially separately authored components, a prominent task (that of the IMs) is to assemble a working pipeline out of the available building blocks. According to the DPs, the fulfilment and assurance of each intent may be distributed across multiple IMs (which are the entities responsible for the intent within a given scope, say, e2e or domain) by means of orchestrating and coordinating multiple closed loops (which are the practical building blocks of software based automation). Finding the right micro-services and chaining them in the right way to form a closed loop needs to happen automatically already during the intent fulfilment phase and should continue during intent assurance. Intent assurance may use the same or potentially changed closed loop chain compared to fulfilment – as assurance may need to onboard components in addition to fulfilment such as real time measurements; whereas assurance may not need components dedicated for one-time service onboarding, only useful during fulfilment.

The interactions of a closed loop with other closed loops and its environment are depicted in Figure 7. A closed loop may receive inputs from its

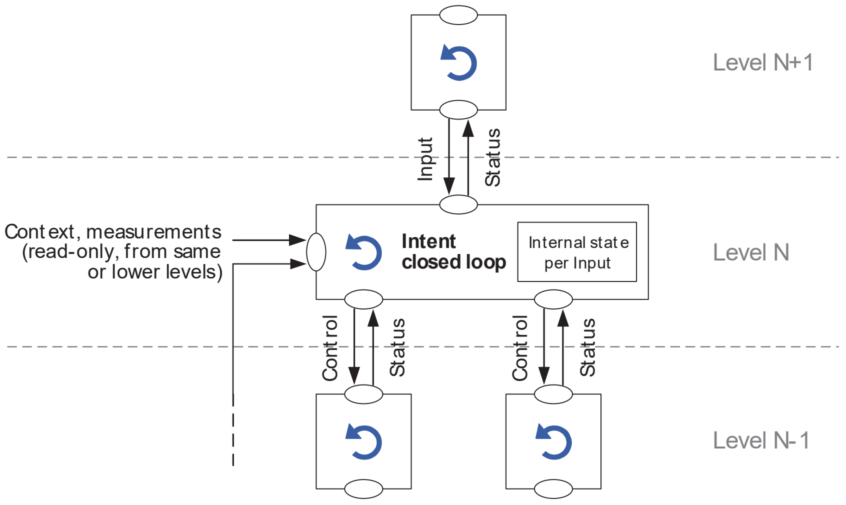

Figure 7 Closed loop automation for intent contextualization and implementation.

hierarchically superior CL. This input is related to the fulfilment or assurance of an intent. Additionally, the CL may collect context and measurements from any other entity, such as domain controllers, or even other CLs. These inputs are used by the CL's internal logic and may be implementation specific. Collecting input from another entity should not have impact on that entity (besides the need for generating the information). On the output side, the CL controls potentially multiple other CLs (at a lower hierarchy level) and provides feedback to its single higher-level CL. As discussed earlier, this status may be an aggregated and transformed version of the status collected from its own controlled CLs. The intent CL implements stateful representation of its inputs as it potentially needs to work with the received information in the long term and has to provide feedback in context of its input later on.

The IM's ability to find the right micro-services automatically imposes certain requirements on the micro-services themselves. First, each closed loop suitable to be a building block in a closed loop declares the following about itself:

- What kind of automation / task it implements (English description: category, scope, etc.) – for human designer and engineering purposes, similarly to API documents and container descriptions.
- Implemented input API (inputs steering the CL's behaviour, e.g., targets, thresholds, operational parameters).

- Data dependencies: data required from other sources for its internal operation.
- Software dependencies, especially other subordinate micro-services implementing data dependencies or analytics services, whose services are used as a dependency.
- Supported APIs on the CL's "S-bound", that is, which other entities N-bound interface this closed loop is able to operate (e.g., the N-bound of a domain controller, SDN-C, or resource manager). The input API of other CLs also belongs to this category, if this CL is positioned, e.g., to be used by an E2E IM.

It was mentioned that, unlike humans, the software systems are unable to evolve their own behaviour while in deployment, and certainly not able to comprehend data or control for which the precise implementation was not provided by their developers. Therefore, two CLs can only be interconnected (i.e., one providing output that is taken as input by the other) if there is a compatible output/input API at the two CLs. That is, intent CLs may be connected based on syntactical and semantical API compatibility. This requires that a CL's input/control API description captures not only its syntax (e.g., "integer is expected") but also its semantic (i.e., what does that integer control in the CL) – otherwise any two interfaces could be potentially connected where one produces an integer and another expects one. Both syntactical and semantical descriptions need to be machine readable. For syntactical descriptions, schemas can be used, which are available in many data modelling languages [16]. For the semantical description of a parameter, the state of the art is to capture the semantics in human readable form (aimed towards human developers) and assign a unique identifier to the parameter (aimed towards the machine). The identifier may be without any context, in which case the machine can evaluate equivalence by exact comparison; alternatively, the ID may be embedded into a hierarchy of standardized or well-known categories such as in Bluetooth Generic Attribute Profile service framework [17] so the ID not only bears the uniqueness of the parameter but also provides semantical description. Note, however, that interpreting the semantic of any parameter by the machine requires that the developer of the software module first understood it and wrote code that causes the software module to behave accordingly – the machine's role in runtime is necessarily limited to conditional checks performed on the IDs to execute the proper subroutine with the code that prescribes the right semantical behaviour.

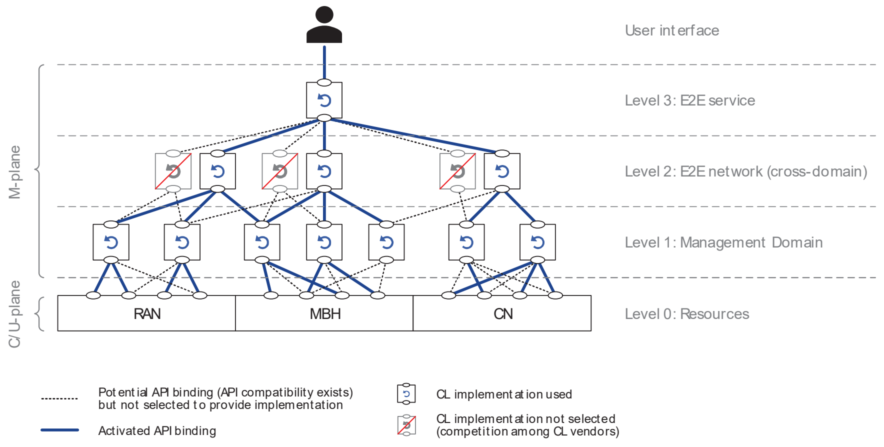
User interface
Level 3: E2E service
Level 2: E2E network (cross-domain)
Level 1: Management Domain
Level 0: Resources
M-plane
C/U-plane
RAN
MBH
CN
Potential API binding (API compatibility exists) but not selected to provide implementation
Activated API binding
CL implementation used
CL implementation not selected (competition among CL vendors)

Figure 8 Closed loop hierarchy for a single intent.

3.2.2 Assembling the CL hierarchy for a single intent (DP#4,5)

During intent fulfilment, the IM responsible for an intent (or part of it, received through delegation from a higher-level IM) has to assemble a CL hierarchy for the intent. Note that with multiple intents being submitted to the system, multiple hierarchies may co-exist, each assembled for their respective intent, and the same CL (that is, micro-service) may be involved in multiple hierarchies (as it will be detailed shortly). Still, when viewing the interworking CLs assembled for a single intent, it shows an upside-down tree topology, as illustrated in Figure 8.

Assembling the interworking CLs may be possible by a breadth-first search process. The search is started by the IM receiving the intent. The top root node of the tree is the closed loop that handles the intent's lifecycle within the IM itself, that is, the logic that implements intent ingestion and decomposition as part of the intent fulfilment phase. This logic then identifies a next level of closed loops by searching for CLs that declare an API compatible with the decomposed and scoped objectives synthetized by the IM's intent lifecycle handler closed loop. Usually the top level IM's decomposition logic works at the e2e service level, and the next level of closed loops will be the business logic of the domain IMs, as it was shown in Figure 3. Alternatively, if the operator provides intent directly to a lower level IM, the search process and the whole tree of CLs is rooted at that IM. In any case, the top-level CL collects all other CLs that could (by means of syntactical and semantical API compatibility) deliver on the decomposed targets. The search then proceeds recursively by transitioning the execution to one of the next CLs and identifying additional CLs, following API compatibility as a means to extend the CL tree. The search terminates if the CL tree is expanded into the domain or resource controller APIs, that is, when the output of every intent related CL is bound to a controller. Technically, at the end of the search process, the resulting graph may not yet be a tree, as multiple higher-level CLs may express compatibility towards the same lower level CL (or domain controller at the lower level), creating diamond shaped links (i.e., when the links from a first level CL branch towards multiple second level CLs, which are then joined at the same third level CL or controller). Such cases represent implementation choices, that is, the intent can be implemented using different sets of CLs. Depending on the implementation of the intent based systems, such choices may be presented back to the operator for manually selecting the preferred implementation (e.g., based on the vendor of the micro-services implementing the CL; or the associated cost or efficiency). Alternatively, the operator may provide policies for the IMs to make selections automatically based on certain conditions, and only those cases not covered by those conditions should be returned for manual inspection. Regardless of the means of resolving the implementation choices, the graph is then purged back to a tree, where the delegation routes and responsibilities between levels of CLs are unambiguous. Note that each CL may in practice require additional supporting micro-services for its own operation, e.g., to assess the real time context of the system, in concert with Figure 7. Those supporting micro-services are not part of the CL tree as they are considered necessary software dependencies for the implementation of the CL. Moreover, such supporting micro-services may be reused by any number of CLs at any level of the hierarchy (as it will be detailed in Section 4), therefore the tree structure that is necessary to avoid control overloads and ambiguity in which entity controls which part of the system does not (and should not) apply to them.

So far, we discussed the assembly of a CL tree that collectively implements the fulfilment and assurance of a single intent. As an intent based system usually handles multiple intents simultaneously, there is a similar tree for each intent. While the per intent graphs are trees, that is, any CL can only occupy a single position in a tree, there is no limitation in any CL participating in multiple trees. The most obvious example is when there is the E2E IM, where a rich implementation may be capable of handling many e2e intents. Alternatively, there may be an IM at a domain level (e.g., RAN) that is able to handle various RAN domain related intents (e.g., related to QoS or efficiency targets). This IM may likely be found to be compatible with and

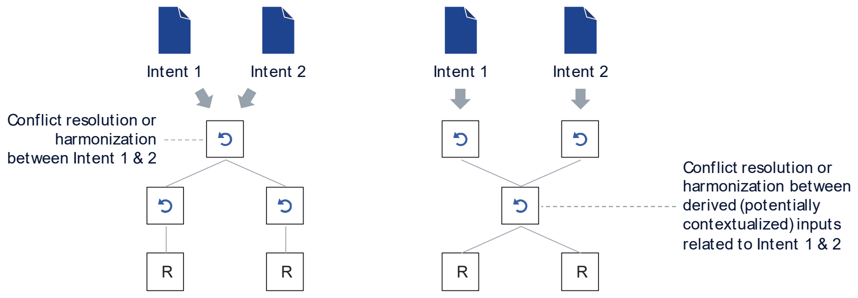
Intent 1
Intent 2
Conflict resolution or harmonization between Intent 1 & 2
R
R
Intent 1
Intent 2
Conflict resolution or harmonization between derived (potentially contextualized) inputs related to Intent 1 & 2
R
R

Figure 9 The same CL participating in the lifecycle of multiple intents.

useful for the implementation of many objectives delegated and scoped to the RAN domain, should such scoping be an action implemented by the E2E IM. This means that the closed loop implementing the intent management logic of this IM would be part of multiple CL trees. These examples are visualized in Figure 9. Note, however, that this figure no longer represents the handling of the same intent, hence the non-tree graph structure. In case a CL is part of multiple CL trees, it is responsible for internally resolving any conflicts within

Splitting the implementation of an intent may not only be based on domains such as RAN, transport or core network, but also based on vendor or software capability areas. That is, parts of the network may be served by different vendor's equipment and network functions, therefore the programmable layer (the available API of their domain controllers) differs. This means that the search process that ultimately builds the CL tree for handling an intent will have to split along service areas, as depicted in Figure 10. Nevertheless, the remaining CL graph is still a tree with no ambiguity in the control and responsibility flow, only the scope of CLs that are specific to handle a particular vendor's domain will be limited to that area.

3.2.3 Auto-generating the interface of the Intent Manager: CL capability aggregation (DP#1)

The Design Principle #1 in Section 3.1 captured that in a realistic intent based system, the expression of intent should not be an unstructured free-style human type of statement (e.g., a natural language) but rather a form of machine interpretable formal language. Moreover, the terms of said language should not be arbitrary but driven by the capabilities of the underlying system so that intents expressed by the operator are indeed comprehensible and interpretable by the system. This goal can be achieved by auto-generating

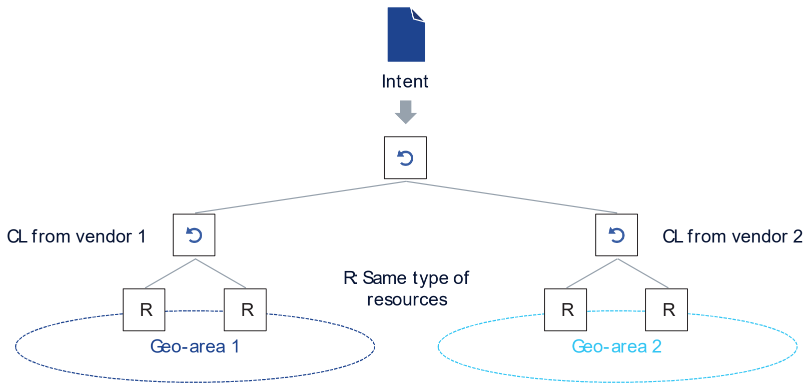

Figure 10 The same intent may be implemented by different CLs in different service areas.

the human-machine intent interface based on aggregating the capabilities of the available CLs and IMs.

The human-machine intent interface can be assembled automatically by exposing the input interface of the CLs which fit into a chain of compatible CLs (from top level to resource/controller level). These APIs present the potential "asks" that could be implemented by the system. Such chains can be discovered statically (that is, based on the exposed input and output APIs as part of the description of the CLs, as discussed in Section 3.2.1) by a search process similar to the one building the CL tree per intent; only this time the search does not know the intent, so it assumes a potential input of a CL as starting point. The assembly of the CL's aggregated capability presents a modular structure: onboarding an intent CL that exposes, e.g., QoS concepts on its N-bound interface and also plugs into a chain of CLs (i.e., has S-bound interface compatible with at least one N-bound interface at the next lower level, etc.) enables the formulation of QoS intents on service level, which then could be offered for the human operator as a plausible intent target. Without such CL in the catalogue of CLs (e.g., repository of micro-service container images), submitting such intent to the system would make no sense as the system lacks the necessary implementation. A modern user interface could create a visual "click through" experience that guides the operator through the potential intents and their parameters.

3.2.4 Additional considerations

Discussing the intent interface and how the supporting implementation may be automatically assembled has implicitly assumed that most intents are related to e2e service, and this assumption may be true for many use cases. Still, an intent based system must not lock down its interfaces towards lower level constructions, as there may be cases when the operator or user of the system may wish or need to interact more directly. First, the operator has to be allowed to exercise control on any level:

- Top-level: for high-level, context-free, declarative intents agnostic to vendor, domain and technology choices. Usual intents include e2e services and their objectives related to QoS or other Service Level Agreement (SLA), which the operator wants to enforce regardless of the current state, load and users of the system. The system is expected to dynamically identify and carry out necessary actions (resource allocations, path reservations, QoS architecture configuration, prioritization, etc.) that is within the capabilities of the traffic facing network functions. Feedback is expected to be on the service level, e.g., what percentage of traffic or customers could be served with the contracted quality, etc. In case of system capacity limits, it is expected that the system signals were, how much and how often the demand exceeds the system capacity so that the operator can schedule upgrades and extensions accordingly.
- Mid-level: e.g., domain level intents that may depend on features implemented only by a specific vendor's network functions, or the intent needs to be scoped to a canary deployment for testing. Providing such intent may require knowledge about detailed network function features and some knowledge on expected results based on manual function parameterization. Still, it is not expected that the intent captures a very specific configuration of network functions, therefore there is room for the automation for discovering the best setup and for continuous context-based adaptation within the approved limits.
- Resource-level: e.g., direct configuration of domain controllers (or even particular network function instances). Such action assumes deep expertise in not only the changed controller or function but understanding of the e2e impacts of administering the changes.

Manually changing low level or local parameters of an otherwise intent based system basically exempts the manually configured entities from being subject of autonomous management. They also have to be treated as boundaries for automation, i.e., values that are given and to which other parts of the system have to adapt. It should be possible to move a manually parameterized set of entities back under intent based management. In any such case, the automation taking over the control should ensure smooth transitions (e.g., only apply automatically derived parameters to newly established user plane connections) so that no disruptive and potentially destabilizing configuration changes are introduced to live user sessions.

4 Self-Learning Closed Loops

Once an intent is fulfilled and all necessary CLs are instantiated that can supervise the enforcement of an intent's objectives, it is necessary that those CLs can adapt the system's configuration according to dynamic changes, events, shift in user demand, load, etc. As intent based systems lack policies that prescribe which actions to execute on what condition, these rules have to be derived autonomously. This section goes deeper into the problem of self-learning, which essentially gives CLs the ability to respond to changes in their execution environment and accumulate knowledge on which action has the highest utility in keeping the intent's objectives satisfied in a given dynamic context.

4.1 Overview

Self-learning closed loops for management automation are part of network and service management standards such as in [ZSM009-3]. Self-learning CLs consider their past actions and their consequences when they make a decision on future actions. This comes from the observation that the utility of a pre-defined action is not absolute; an action that is very useful in some cases may be not effective or even counter-productive in other cases. Supposing that a CL with a set of objectives may choose from a list of actions through which it may impact the network and services, a self-learning CL should discover which is the best action under a given context that maximizes its utility and reach the objectives. Such self-learning capability depends on well-defined metrics that quantify the utility of the action (within a given context). Currently such metrics have not been standardized or even de-facto defined, creating a gap between this concept and its potential implementation.

A follow-up technical problem of self-learning CLs is related to the collection of any such utility metric (once it is defined). It cannot be assumed that the CL itself (which is an automation logic) is able to operate in a standalone way, generating all the measurements needed for its decision process on its own. That would not only mean that the self-learning CL has to have access to low-level resource layers and at the same time be present in many technology domains (i.e., essentially being a single central entity with connections to

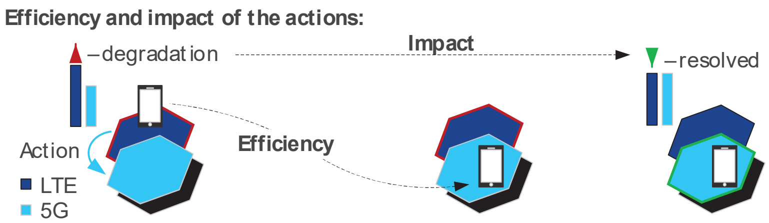

Figure 11 The concept of efficiency and impact in ZSM009-3.

every other domain and resource controller), but also that multiple self-learning CLs (e.g., from different vendors) would need to implement similar or, for interoperability, the exact same utility measurements on a variety of input data (where the data itself may also come from multiple vendors). Due to the lack of standards, there is currently no accepted architecture that would let self-learning CLs focus on their analytics and automation logic without having to implement their own utility measurements.

A key concept related to self-learning CLs is the self-monitoring of automated actions triggered by CL itself. The self-monitoring may be defined by the efficiency and impact related to the action as illustrated in Figure 11 and explained below [15].

Definition of efficiency (as per the current [15]): the requested action is successfully completed with the specific scope and target such as User Equipment (UE), cell, etc. A self-learning in CL is supposed to filter out actions that are inefficient in a given context. Example: successful execution of vertical handovers due to the action of reconfiguring radio thresholds is an efficient action (because the handovers do happen as a result of the reconfiguration). If some UEs were to reject the handover (e.g., due to lack of capability, or lack of real coverage from the targeted frequency layer), the action would be less efficient or not efficient at all.

Definition of impact (as per the current [15]): the action is impactful if it has reached its goal (e.g., resolved a system state degradation). A self-learning in CL is supposed to filter out actions that have low impact in a given context. Example: if the vertical handovers have resolved the degradation by decreasing the load in the resource layers, the action was impactful. Otherwise, even if the action was efficient (i.e., it did successfully change what it was designed to change) the impact is low or even negative.

While the above concept of efficiency and impact was accepted in [15], there is no detail about how to compute or quantify the efficiency and impact of any particular action. The challenge is that a uniform metric would be preferred to enable the comparison of efficiency and impact across different actions, yet the difference in the actions themselves and in the objective of the self-learning CLs (i.e., what to optimize for) mean that the calculation of efficiency and impact has to be specific to the action and to the CL's logic itself. Therefore, the general concept of efficiency and impact-based evaluation cannot be practically used without inventing further methods to calculate and report efficiency and impact type of metrics that are suitable for self-learning CLs.

4.2 Proposed Operation

This section proposes a micro-service-based architecture decomposition and related metadata and interface specification to enable standard realization and automated assembly of self-learning CLs. A self-learning CL should be decomposed into interworking micro-services where one or more micro-services are responsible for action utility measurements and one or more micro-services are responsible for automation logic (Figure 12). An automation logic micro-service may consume measurements generated by a measurement micro-service and produce decisions and actions. Additionally, an automation logic micro-service may produce insight that is consumed by one or more other automation logic micro-services. All these automation logic micro-services together with the utility measurement micro-services may collaboratively implement a complex distributed automation task, which is the business logic of the self-learning CL.

Measurement micro-services are reusable components potentially providing input to multiple unrelated automation logic micro-services in separate self-learning CLs. Each measurement micro-service is specialized to evaluate network, service, traffic, user, equipment or any other network/user entity state or performance from a specific perspective and provide a standardized utility metric to any interested automation logic micro-service. The standard utility metric is proposed to be the so-called incident rate. Incident rate is a metric defined as a function of (1) a number of events that are considered negative and (2) a total number of events. For example, the function may produce the ratio of (1)/(2). The definition of an event and when it is considered negative is defined by the implementation of the measurement micro-service.

Each measurement service has, besides its code implementation, three attributes: (1) objective; (2) input domain; (3) scope; defined as follows.

Objective: Any particular measurement micro-service is implemented to generate incidents compatible with a specific objective (e.g., SLA assurance

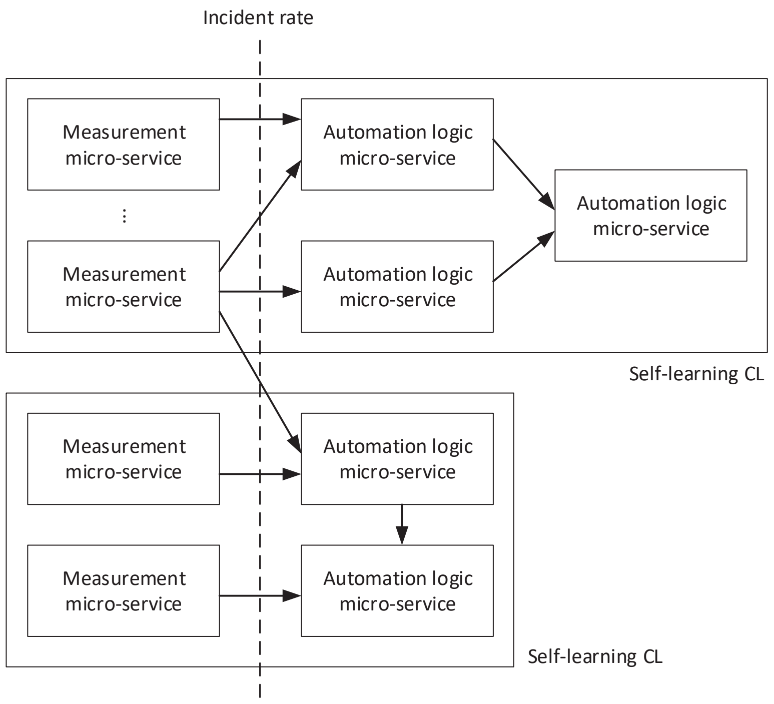
Incident rate
Measurement micro-service
Measurement micro-service
Automation logic micro-service
Automation logic micro-service
Automation logic micro-service
Self-learning CL
Measurement micro-service
Measurement micro-service
Automation logic micro-service
Automation logic micro-service
Self-learning CL

Figure 12 Micro-service based self-learning CL architecture.

objective requires incident definition related to the quality of the service; power efficiency objective requires incident definition that is related to the power consumption of the services; etc.). The objective is a static (design time) metadata that is part of the micro-service implementation.

Input domain: The implementation of the measurement micro-service also defines the input domain from which the incident measurements are generated (e.g., Network Function (NF) PM counters of a specific network function type such as gNB-CU, cloud infra Virtual Machine (VM) resource counters, Virtual Network Function (VNF) application metrics, etc.). The input domain is a static (design time) metadata that is part of the micro-service implementation.

Scope: The scope of a measurement micro-service is defined dynamically when the measurement micro-service is deployed in a system (e.g., by an orchestrator management function). The scope of a measurement micro-service is the intersection of its input domain and its deployment area (e.g.,

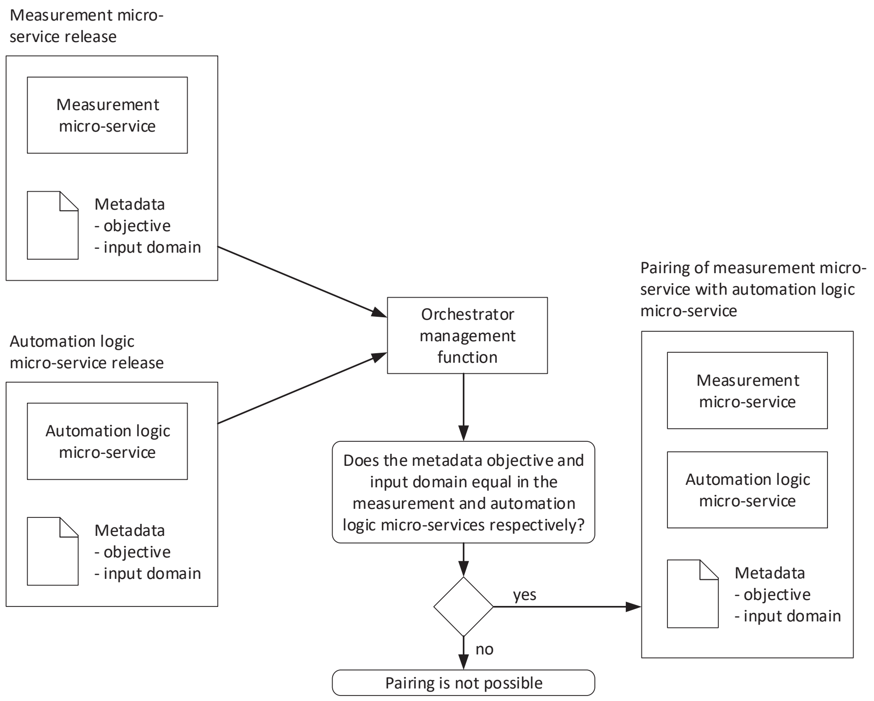
Measurement micro-service release
Measurement micro-service
Metadata
- objective
- input domain
Automation logic micro-service release
Automation logic micro-service
Metadata
- objective
- input domain
Orchestrator management function
Does the metadata objective and input domain equal in the measurement and automation logic micro-services respectively?
yes
no
Pairing is not possible
Pairing of measurement micro-service with automation logic micro-service
Measurement micro-service
Automation logic micro-service
Metadata
- objective
- input domain

Figure 13 Pairing of a measurement micro-service with an automation logic micro-service based on micro-service metadata.

a geo-area, a network slice, etc.). That is, a deployed measurement micro-service will generate incidents from the intersection of its input domain and the deployment area (e.g., for all gNB-CUs within a network slice) where the generated incidents are compatible with the measurement micro-service's objective (e.g., SLA assurance).

An automation logic micro-service autonomously makes decisions and actions to converge a network or service towards a specific objective. In order to make the automation logic micro-service part of a self-learning CL, it should be coupled with the right measurement micro-service that generates incidents based on the same objective. This requires that automation logic micro-services are also released with metadata defining their objective and input domain, similarly to measurement micro-services. This enables an orchestrator management function to automatically pair automation logic micro-services with compatible measurement micro-services (based on matching objective and input domain in their metadata), assembling the self-learning CL from the disaggregated micro-services.

4.3 Implementation and Benefits

The loose coupling between the measurement micro-services and automation logic micro-services promotes an extendable marketplace where a growing variety of incident measurements can be implemented and published by different measurement micro-services (by a multitude of vendors), but all exposing a uniform API (the incident metric) towards the automation logic micro-services (by the same or different vendors).

The deployment of a self-learning CL is performed by an orchestrator management function. The prerequisite for the deployment is to have all micro-service implementations and attached metadata (both for measurement micro-services and automation logic micro-services) available to the orchestrator function. (Availability may be satisfied by the ability to acquire such data on-demand, e.g., from a software store, not necessarily by having everything on-premise.)

The deployment of a self-learning CL commences on a request coming from an external entity above the orchestrator function (e.g., from the operator). The trigger should identify the CL and the deployment scope. The CL is represented by the list of automation logic micro-services, wherein each automation logic micro-service is represented by its implementation and the objective and input domain metadata (Figure 14).

The first step of the self-learning CL deployment (Figure 15) is to collect the measurement micro-service dependencies of the self-learning CL, i.e., the list of measurement logic micro-services that are needed to supply the

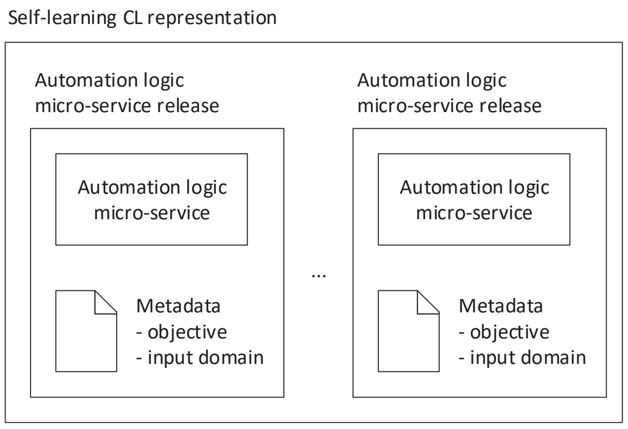
Self-learning CL representation
Automation logic micro-service release
Automation logic micro-service
Metadata
- objective
- input domain
...
Automation logic micro-service release
Automation logic micro-service
Metadata
- objective
- input domain

Figure 14 Self-learning CL representation for deployment.

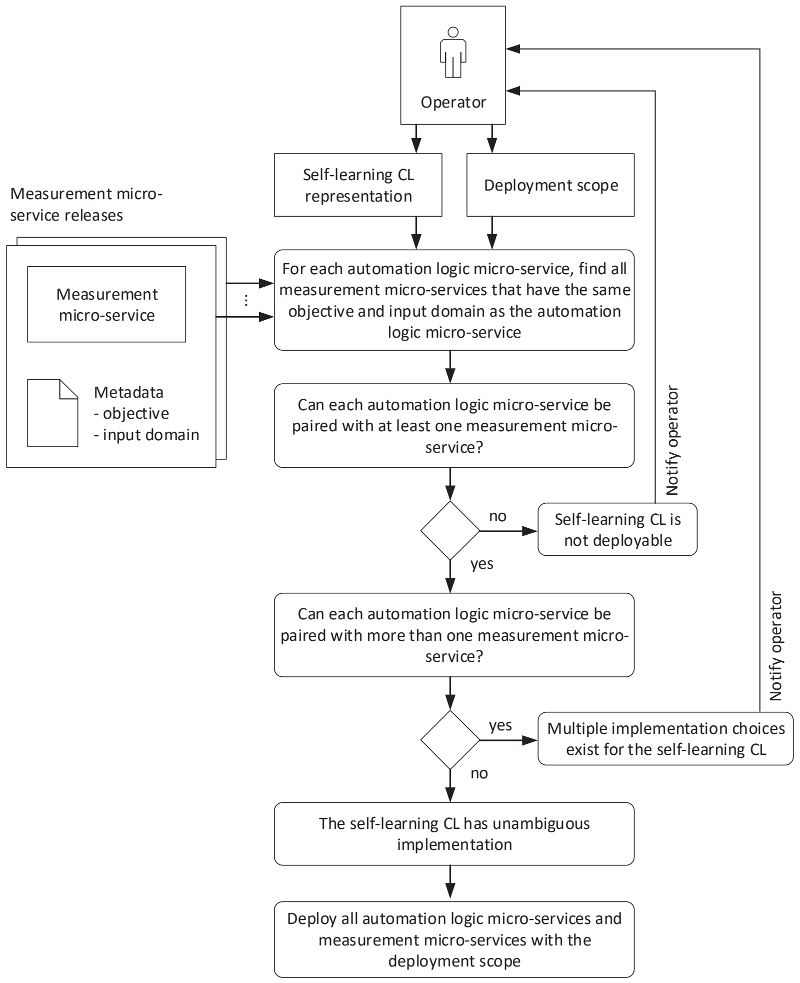
Operator
Self-learning CL representation
Deployment scope
Measurement micro-service releases
Measurement micro-service
Metadata
- objective
- input domain
For each automation logic micro-service, find all measurement micro-services that have the same objective and input domain as the automation logic micro-service
Can each automation logic micro-service be paired with at least one measurement micro-service?
no
yes
Self-learning CL is not deployable
Notify operator
Can each automation logic micro-service be paired with more than one measurement micro-service?
yes
no
Multiple implementation choices exist for the self-learning CL
Notify operator
The self-learning CL has unambiguous implementation
Deploy all automation logic micro-services and measurement micro-services with the deployment scope

Figure 15 Method steps of the orchestrator management function during the deployment of a self-learning CL.

incident rate to the CL's automation logic micro-services. A measurement logic micro-service is able to supply the incident rate to an automation logic micro-service if the objective and input domain of the two micro-services are the same. The right measurement micro-service for a given automation logic micro-service may be found by searching through all known measurement micro-services and comparing the metadata of each measurement micro-service with that of the automation logic micro-service. On multiple potential matches, the initiator of the deployment (e.g., the operator) may be presented with a choice option so that it can select the preferred implementation. Note that the scope of the two micro-services are aligned by the orchestration function by deploying them to the same area. The means for this is out of scope of this article as it may involve state of the art software deployment and database integration steps.

4.4 AI/ML for Automation

Self-learning capabilities are often attributed to using Artificial Intelligence (AI) or, in practice, ML techniques. Using ML to implement self-learning is indeed feasible, provided that the ML specific aspects of closed loops are considered. This section intends to touch on a few key technical questions or challenges related to embedding ML based functions in intent based systems.

4.4.1 Data management aspects

The use of ML methods starts with defining the data types and data sources upon which selected algorithms are trained and later, once a model is trained, executed. Considering service and network management use cases, key data types and data sources could be the followings:

- Network measurements from telco software, i.e., the application part of VNFs and Cloud-native Network Functions (CNF) (related to 3GPP standard or proprietary functions in RAN, transport, core, edge, Multi-access Edge Computing (MEC), Radio Intelligent Controller (RIC), etc.). Traditionally, these are the evolved versions of what is traditionally configuration management (CM) for configuration data and PM/FM for operational or runtime data. This type of data is suitable for numerical multi-dimensional time series analytics [11, 12]. While standardized metrics [18] continue to be provided by modern software implementations, the rise of virtualization and micro-service architecture also transformed logging practices. Currently kernel, system and application text logs, originally written by engineers to be read by other engineers, start to become a rich set of insight about how telco software operates. The machine-based analysis of this type of data requires using advanced ML techniques beyond statistical evaluation such as log analytics [19] to deliver actionable insights for closed loop automation.
- Application insight from devices and apps (mobile Internet apps, V2X modem, Industry 4.0 robot, xR, etc.) and their correlation with network insight has high potential to implement feedback loops based on the

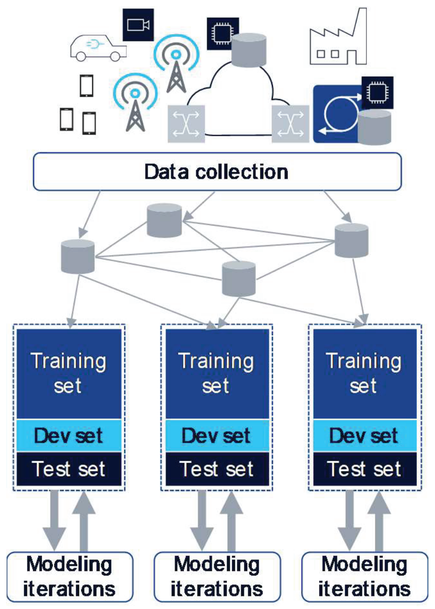

Figure 16 Data collection and training instrumentation is key for ML based automation.

actual service the network provides to the ultimate end users. Assuring true e2e SLAs, for example, is only possible if reliable and timely measurements are available from the very end of the e2e. The correlation with network insight is necessary to transform end user perception into potential network side actions executable by domain controllers, such as knowing that increasing a bandwidth allocation on a particular transport service will convert to better service quality in an e2e service rather than burning more resources by providing more speed to services where it is not recognized.

- Cloud infra measurements from the on-premise or public cloud infrastructure hosting the network functions. With cloud native becoming the new standard in network function architecting, additional dimensions of operation such as the state of the supporting infrastructure and cloud orchestration actions have to be considered by network and service automation.

Collecting large amount of diverse data from a distributed system requires data collection framework that is light on data sources (i.e., every piece data is produced only once by a source), efficient in data routing and replication

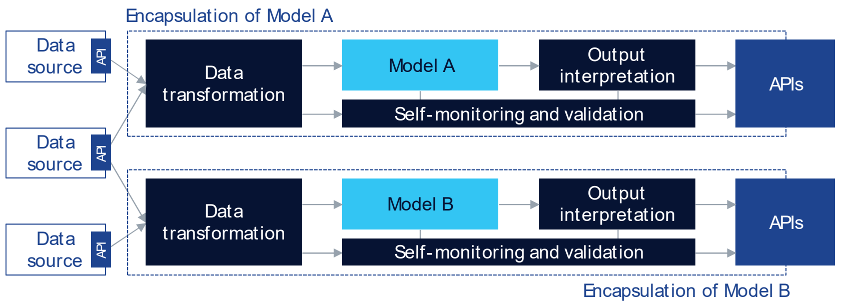

Figure 17 Embedding ML models for encapsulating technology specific details.

towards multiple data consumers, and allows for programmable lookup, subscription or retrieval of available or new data [20]. In addition to identifying the data sources and collecting the right type and amount of data, developing ML models require properly setup data sets (training, development, test) and compute framework for model training and validation.

4.4.2 Encapsulation of ML models

Using ML models in software usually does not mean that the ML model's input and output are directly exposed as an API. Instead, there is additional software encapsulation (Figure 17) that embeds the model for various reasons:

- Easier deployment: hide vendor specific SW stack and dependencies, e.g., by using state of the art container technology.
- Interfacing with input data sources: on the input side, implement data source specific APIs to receive or collect data, and transform it to produce the exact data format (syntactically and semantically) that is expected by the model.
- Interfacing with published APIs: on the output side, transform the model's raw output to the representation that conforms to the APIs published towards other entities.
- Self-monitoring and self-validation: implement logic that is able to detect if the model is not producing reliable output for various reasons detailed shortly. Additional logic may be embedded to trigger re-training, model update, or fallback to non-ML based legacy business logic in case the currently operated model is unreliable. Such actions require integration with a model management or ML orchestrator framework [21].

4.4.3 Deviation between training and field data

The state-of-the-art technology of ML depends on model training and applying trained model to new data for inference purposes. Regardless of whether model training and inference based on trained model are executed as strictly separate phases (e.g., as common with supervised training of deep learning models) or they partly or fully overlap (e.g., with in-situ reinforcement learning), the latest model that is applied to fresh field data bears the risk of not producing reliable output. This is due to the nature of ML technology: models are derived through a complex training process based on past data, with no analytical description available of how the model will perform on unseen data. While there are several techniques and best practices to organize training data in a way that its distribution approximates that of the data to be analysed after deployment, there is no guarantee that live data will not produce such outliers on the input layer of an ML model that throws the model off its boundaries. Unlike mathematical formula based numerical algorithms where proof can be derived on at least boundaries of the algorithm's output, generally no such constraints are available for ML models. Therefore, it is even more important that ML models are surrounded with additional logic that implements safeguards both by checking the input data before it is processed by a ML model and by validating the output of the model before taking it forward as input for further analytics or decision making.

Common reasons why operational or field data may be substantially different from the data on which a model was trained include the followings:

- Changes in the network load due to activities not related to communication itself
- Changes in the behaviour of users or applications that are generating the demand, resulting in different traffic patterns and ultimately different network states
- Changes in the network itself, such as evolving technology and deployment
- Changes in the NF software releases and deployment (upgrading, scaling, etc.)

Some of the others and when they happen may be well known by the operator of the network (e.g., making changes to the network, rolling out new software releases, etc. are certainly something that could be tracked). Others have to be discovered retrospectively by observing their implicit impact on network and service KPIs. In any case, continuous monitoring and verification of training data vs. field data and the reliability of deployed ML models is required. Preferably, such monitoring should deliver the following insights, which could even be channelized to automated actions, such as model re-training, downgrade or fallback to non-ML solutions, as mentioned above:

- Detect that there is an issue with the reliability or accuracy of an ML model (i.e., separating this root cause from others such as there is a degradation in the network itself due to wrong configuration and as a side effect that also causes poor ML accuracy).
- Early detection of mismatching the distribution of training data (on which the model should perform well) and the actual data subject to analysis by the model. Operating on patterns, types or quality of data that was completely missing from the training set of a model is a common reason for model accuracy degradation, therefore implementing automated checks on the data level may be a useful defense layer.
- Proactively prepare for model re-training by keeping batches of new data in cache. This practically means to have a ring buffer holding an amount of the latest data that could be promoted into the model's training data set, should it be detected that new patterns contained in the latest data is the reason behind poor model performance. The amount of data to be cached depends on how much data the ML pipeline usually requires generating a solid model. This can jump-start the re-training process by saving time that would otherwise spent on potentially days or weeks of data collection; additionally, if the new patterns on which the model failed are scarce, it may take a long time before those pattern re-appear again so that they can be caught.

The consequences of model accuracy degradation coupled with closed-loop automation may be quite drastic, even leading to "spectacular failures". These are non-intuitive failures that, as opposed to the understandably poor output when applying the model on unexpected data, happen when the model processes data that is seemingly (to a human) similar to the data on which the model was trained. ML models may fail in non-intuitive and non-graceful ways because an ε change in the model input may trigger arbitrarily large (unbounded) change in the model output if unbound activation functions are used, such as the popular ReLU [22, 23]. This attribute creates a new type of bugs (in addition to conventional software/implementation bugs like null pointer dereference, etc.) and opens up new adversary attack vectors [24]. Therefore, in systems depending on ML for high level or automation, new mitigation techniques are required to harden ML technology or the surrounding software layers against such failures.

5 Conclusion

Intent based management is a new paradigm in network and service management. In this article, we have looked through a brief history of intent in telecommunications and observed how humans have used intent successfully to understand the two key enablers for successful intents: intelligence and shared context between the one creating the intent and the one interpreting and implementing it. In intent based networking, this means intelligence in the network, hence the term intelligent intent based networks or I^2BN. This article discussed an intent framework where intelligence comes from self-learning closed loops, and shared context is achieved by expressing intents by means of aggregated network capabilities rather than arbitrary terms. Self-learning closed loops are necessary to enable automation with adaptation capabilities; instead of simply replaying pre-programmed workflows, they can synthetize new sequences of actions as a response to dynamically changing user demand, network state and resource load. This requires stateful representation and memory of which actions were triggered in which contexts and what was their outcome in terms of efficiency and impact. The article proposed concrete means to quantify and unify learning feedback across various closed loops that are potentially implemented by different vendors. The other contribution of the article is to approach the design of the human-network intent interface from a novel direction. Recognizing the difference between human and artificial intelligence, it is more pragmatic to present the human user of the system with potential automation capabilities exposed by available closed loop implementations, rather than presenting the network with a generic human expression or language describing intents fully independently from software APIs. The design, architecture and implementation aspects of intents are discussed in the context of the ZSM architecture and closed loop automation advanced topics. The technical consequences of using machine learning for implementing parts of the intent based system are also considered. Best practices to package and distribute micro-services with ML were reviewed, as well as potential challenges with data collection, detection of model obsolescence and non-intuitive model failures. The findings of the article could be channelized into future standardization streams in ZSM such as the new ZSM011 Intent-driven autonomous networks and ZSM012 Enablers for Artificial Intelligence-based Network and Service Automation.

Acknowledgement

This work was supported by the National Research, Development and Innovation Fund of Hungary in the frame of FIEK_16-1-2016-0007 (Higher Education and Industrial Cooperation Center) project.

References

[1] J. C. Strassner, Policy-Based Network Management, Elsevier, 2003.
[2] J. C. Strassner, "Intent-based Policy Management," in *SDNrg, IETF 95*, Buenos Aires, 2016.
[3] A. Clemm, L. Ciavaglia, L. Granville and J. Tantsura, *Intent-Based Networking – Concepts and Definitions,* IETF Network Working Group, 2020.
[4] C. Li, O. Havel, W. Liu, A. Olariu, P. Martinez-Julia, J. Nobre and D. Lopez, " Intent Classification," IRTF, 2020.
[5] 3GPP TR 28.812, "Telecommunication management; Study on scenarios for Intent driven management services for mobile networks," 2020.
[6] 3GPP TS 28.312, "Management and orchestration; Intent driven management services for mobile networks," 2020.
[7] ONF TR-523, "Intent NBI – Definition and Principles," 2016.
[8] R. Addad, D. Dutra, M. Bagaa, T. Taleb, H. Flinck and M. Namane, "Benchmarking the ONOS Intent Interfaces to Ease 5G Service Management," in *IEEE Global Communications Conference (GLOBECOM)*, Abu Dhabi, 2018.
[9] ETSI ZSM 005 V0.4.1, "Zero-touch network and Service Management (ZSM); Means of Automation," 2020-03.
[10] ETSI ZSM 002 V1.1.1, "Zero-touch network and Service Management (ZSM); Reference Architecture," 2019-08.
[11] P. Szilágyi and S. Nováczki, "An Automatic Detection and Diagnosis Framework for Mobile Communication Systems," *IEEE Transactions on Network and Service Management*, vol. 9, no. 2, pp. 184–197, 2012.
[12] G. Ciocarlie, U. Lindqvist, K. Nitz, S. Nováczki and H. Sanneck, "DCAD: Dynamic Cell Anomaly Detection for operational cellular networks," in *IEEE Network Operations and Management Symposium (NOMS)*, Krakow, 2014.

[13] P. Ramachandra, K. Zetterberg, F. Gunnarsson, R. Moosavi, S. B. Redhwan and S. Engström, "Automatic neighbor relations (ANR) in 3GPP NR," in *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, Barcelona, 2018.

[14] ETSI ZSM 009-1 V0.10.4, "Zero-touch network and Service Management (ZSM); Closed Loop Automation – Enablers," 2020-12.

[15] ETSI ZSM 009-3 V0.1.0, "Zero-touch network and Service Management (ZSM); Closed Loop Automation – Advanced Topics," 2020-10.

[16] A. Agocs and J. L. Goff, "A web service based on RESTful API and JSON Schema/JSON Meta Schema to construct knowledge graphs," in *International Conference on Computer, Information and Telecommunication Systems (CITS)*, Colmar, 2018.

[17] Bluetooth SIG, "Bluetooth Core Specification v5.2," 2019.

[18] 3GPP TS 28.552, "Management and orchestration; 5G performance measurements," 2020.

[19] M. Sun, G. Convertino and M. Detweiler, "Designing a Unified Cloud Log Analytics Platform," in *International Conference on Collaboration Technologies and Systems (CTS)*, Orlando, 2016.

[20] 3GPP TR 23.700-91, "Study on enablers for network automation for the 5G System (5GS); Phase 2," 2020.

[21] ITU-T, "FG ML5G – Unified architecture for machine learning in 5G and future networks," 2019.

[22] H. Liu, Z. Zhou, F. Shang, X. Qi, Y. Liu and L. Jiao, *Boosting Gradient for White-Box Adversarial Attacks*, arXiv, 2020.

[23] C. Xu, D. Li and M. Yang, *Improve Adversarial Robustness via Weight Penalization on Classification Layer,* arXiv, 2020.

[24] Y. Shi, Y. E. Sagduyu, T. Erpek and M. C. Gursoy, *How to Attack and Defend 5G Radio Access Network Slicing with Reinforcement Learning,* arXiv, 2021.

Biography

Péter Szilágyi received an M.Sc. degree in Software Engineering from the Budapest University of Technology and Economics (BME), Hungary, in 2009. He is a senior research engineer and DMTS at Nokia Bell Labs. His research history includes telecommunications and software, artificial intelligence, data analytics, 5G and verticals, connected vehicles, end-to-end system engineering and rapid prototyping. He has more than 40 patents and 25 publications in book chapters, journals and conferences.