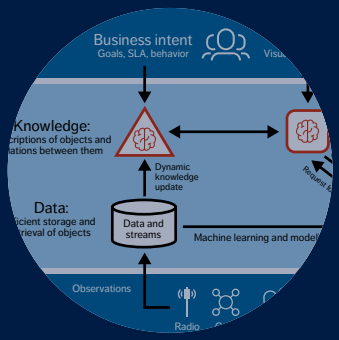
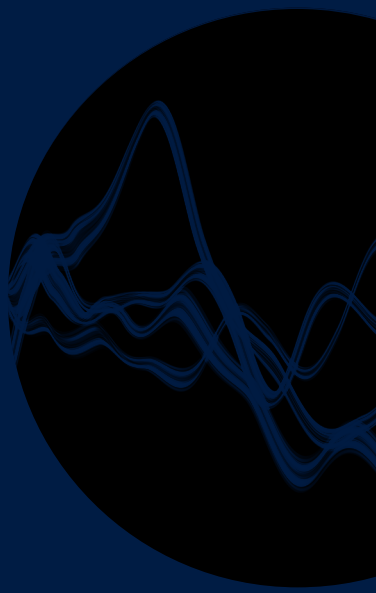
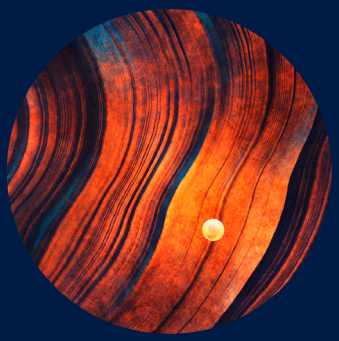
CHARTING THE FUTURE OF INNOVATION | #11·2020

Ericsson Technology Review





ADAPTIVE INTENT-BASED NETWORKING



ERICSSON

Cognitive processes for adaptive intent-based networking

Autonomously operated and self-adapting networks will make it possible to utilize the capabilities of 5G networks in new business models and achieve an unprecedented level of efficiency in service delivery. Intents will play a critical role in achieving this zero-touch vision, serving as the mechanism that formally specifies what the autonomous system is expected to do.

JÖRG NIEMÖLLER, LEONID MOKRUSHIN, SWARUP KUMAR MOHALIK, MARTHA VLACHOU-KONCHYLAKI, GEORGE SARMONIKAS

5G networks introduce unprecedented flexibility and dynamic adaptation into service delivery and network resource utilization. In the business layer, this is reflected in the ability to offer customizable service products with detailed agreements on functional and non-functional characteristics as well as fast delivery. Dynamic adaptation to changes within the constraints of stringent requirements on lead and reaction times is beyond the capacity of a human workforce. Extensive automation will be necessary to overcome this challenge.

■ The zero-touch paradigm implies that the operation of services and the underlying networks is autonomous and does not require human intervention. To achieve this, the zero-touch system must be able to handle the complexity caused by continuous changes to the system at the same time that it delivers services to users and manages issues such as the cost of resources versus the budget available, the legal compliance of the service and the security of the setup. This is challenging for technical systems in real-world scenarios.

For example, successful service operation requires each service to be properly provisioned and assured to deliver the promised function with agreed

performance metrics. Complexity arises as a result of changes to contracts, products, customer preferences, the business strategy or the environment in which the service is being offered. Users may start exhibiting new behavior, leading to varying service usage patterns and network loads, or the network may change due to upgrades, reconfiguration or outages. Some changes may be regular and predictable, whereas others are sudden and surprising.

To manage all of these concerns autonomously and adapt its behavior appropriately, a zero-touch system must understand every aspect of what is expected of it. Each requirement and goal must be carefully defined in order for technical processes to derive suitable and optimized actions to manage it. These definitions are known as intents.

From the perspective of a human operator, an intent expresses the expectation of what the operational system is supposed to deliver and how it behaves. In light of this, we define intent as "formal specification of all expectations including requirements, goals and constraints given to a technical system."

The role of intents in cognitive networks

Everything an autonomous system needs to know about its goals and expected behavior must be defined with intents. The system will not perform any operation unless it relates to the fulfillment and assurance of an intent, which means that all goals – including those that may have been considered "common sense" in human-operated systems – must be expressed as intents.

An intent in an autonomous system is ideally expressed declaratively – that is, as a utility-level goal that describes the properties of a satisfactory outcome rather than prescribing a specific solution. This gives the system the flexibility to explore various solution options and find the optimal one. It also allows the system to optimize by choosing its own goals that maximize utility.

Unlike traditional software systems, where requirements are analyzed offline to detect and resolve conflicts prior to implementation, intents are added to an autonomous system during runtime. Adaptation to changed intent as well as conflict detection and resolution are therefore essential capabilities of an autonomous system.

One of the benefits of expressing intents as utility-level goals is that it helps the system cope with the conflicting objectives of multiple intents. This is vital, because an autonomous system often has to take multiple intents into account before making a decision.

For example, an autonomous system may have one intent to deliver a service with high QoE, while another may be to minimize resource spending. It can resolve such conflicts either explicitly from weights that introduce relative importance or implicitly from properties of preferential outcomes as defined in utility-level goals.

Expectations originate from contracts or business strategy and remain constant when the underlying system is replaced or modified. Consequently, when setting up the intents, it is important that they are formulated in an infrastructure-agnostic way, so that they can be transferred across system generations and implementations.

Terms and abbreviations

AI – Artificial Intelligence | **BSS** – Business Support Systems | **CEM** – Customer Experience Management | **IoT** – Internet of Things | **KPI** – Key Performance Indicator | **OSS** – Operations Support Systems | **RDF** – Resource Description Framework | **SLA** – Service Level Agreement | **SLO** – Service Level Objective | **SLS** – Service Level Specification | **SON** – Self-Organizing Networks | **TOSCA** – Topology and Orchestration Specification for Cloud Applications | **VNF** – Virtual Network Function

In short, the intent establishes a universal mechanism for defining expectations for different layers of network operation. It expresses goals, utility, requirements and constraints. It defines expectations on service delivery as well as the behavior of the autonomous operational system and the underlying network.

> [THE INTENT] EXPRESSES GOALS, UTILITY, REQUIREMENTS AND CONSTRAINTS

Service-specific intents

One essential type of intent relates to the specification of services. Service-specific intents state expected functional and performance characteristics. Service Level Agreements (SLAs), Service Level Specifications (SLSs), Service Level Objectives (SLOs) and TOSCA (Topology and Orchestration Specification for Cloud Applications) [1] models are all examples of service-specific intents that are used on different levels in the operations stack.

SLAs are business support systems (BSS) objects. Service-specific intents based on SLAs specify the promised service and include expected performance details and business consequences such as payment for delivery and penalties when failing.

SLSs/SLOs define the service delivery details at operations support systems (OSS) level. Based on this input, autonomous OSS would plan detailed tasks to realize the service delivery. TOSCA models would be used to express further technical details the OSS generate (expectations from orchestration and assurance).

In multiple stages, the autonomous operation makes decisions about further details. Higher-level intent is the input leading to the lower-level intent that is used to distribute specific goals to subsystems. For example, SLAs/SLSs are the intents that express a terminal goal of the OSS. The OSS then decide which TOSCA model would be the best option to deliver the promised performance with minimal resource usage. The selected TOSCA model is the instrumental goal of OSS and becomes an intent and terminal goal for the orchestrator.

This pattern of making decisions based on a given intent and taking action by sending lower-level intents to subsystems is the key interwork mechanism of intent-based operation, according to which the entire operations stack of autonomous networks is built.

Strategic and behavioral intents

Beyond all the service-specific intents that an autonomous system must have, it also requires guidance on how to handle strategic and behavioral concerns. Traditionally implemented in the form of manually coded policies, this type of guidance steers general system behavior and supports the type of decision-making that has traditionally been based on human intuition and experience, along with knowledge about context and operator strategy. Intent-based operation makes it possible for operators that want to handle these concerns in a more dynamic fashion to replace manually coded policies with strategic and behavioral intents.

This is useful in cases where the operator chooses to require a default minimum security level that differs from that which is implemented into the service, for example. In these cases, dedicated intent can be used to set the security level for all services that do not specify it directly.

With regard to legal compliance, services may be delivered in multiple markets where different rules apply. A legal-compliance intent requires compliance and potentially specifies the details.

Since there is always a risk of service degradation when changes are initiated and risky actions may sometimes lead to a higher margin, risk-management intents can be used to convey how the operator wants the autonomous system to balance risks versus potential gain.

Reporting/escalation intents steer how the autonomous system interacts with the human workforce by reporting progress status on intent fulfillment and seeking manual decision in escalations.

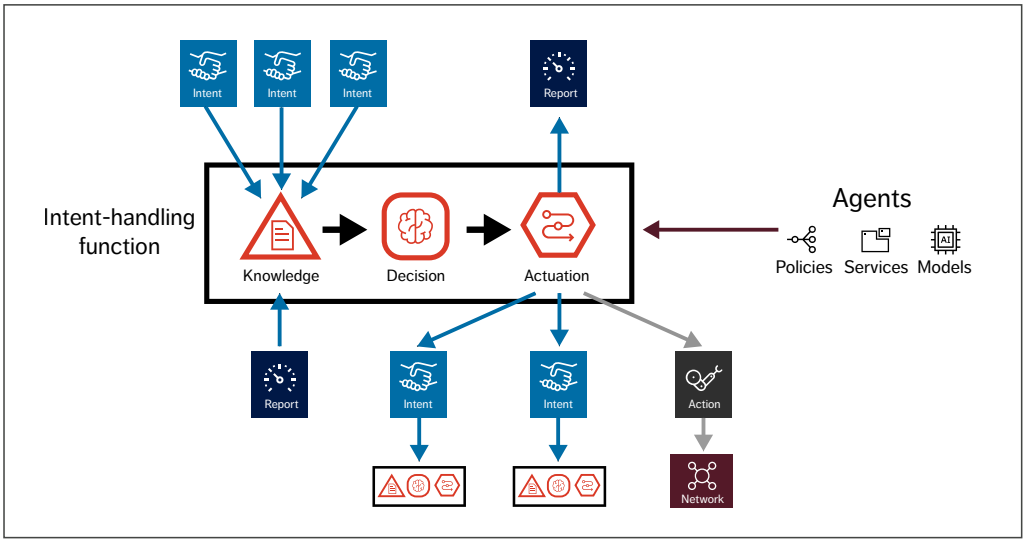


Figure 1 The intent-handling function

Formally expressing an intent

An autonomous system requires intents to be formally defined in a machine-readable and processable way, but the broad range of considerations involved and their abstract semantics are often difficult to structure. Techniques from knowledge management and semantic modeling enable the creation of an ontology of intent, based on an extensible metamodel. Resource Description Framework (RDF) [2] and RDF Schema [3] standards can be used for knowledge modeling.

Technical functions such as contract and order management would directly use RDF objects to communicate intent. Intent specified directly by human operators would require an intuitive frontend, potentially using natural language.

Intent handling

The operation of services within an intent-based network also requires the introduction of intent-handling functions in the operations stack and functional architecture. An intent- handling function receives the intents, decides which actions must be taken to optimally fulfill all given intents and implements its decisions.

Intent-handling functions have a knowledge base that contains the intent ontology. They also have machine-reasoning capabilities to realize knowledge-driven decision-making processes.

Machine reasoning plays a key role in intent handling, with its capability to understand abstract concepts from diverse domains and provide precise, specialized conclusions based on precedent and observation. Probabilistic modeling contributes quantification of risk and uncertainty, which is essential to make informed decisions when facing conflicting goals and new situations.

Figure 1 shows how the intent-handling function works. While its implementation is domain-specific, its interface is generic. It receives intents that express all types of expectations. It is equipped with policies and artificial intelligence (AI) models that implement the capabilities needed for analyzing the system state and finding optimized operational actions based on observations from the operated environment. The intent handler also reports the fulfillment and assurance status of its intents.

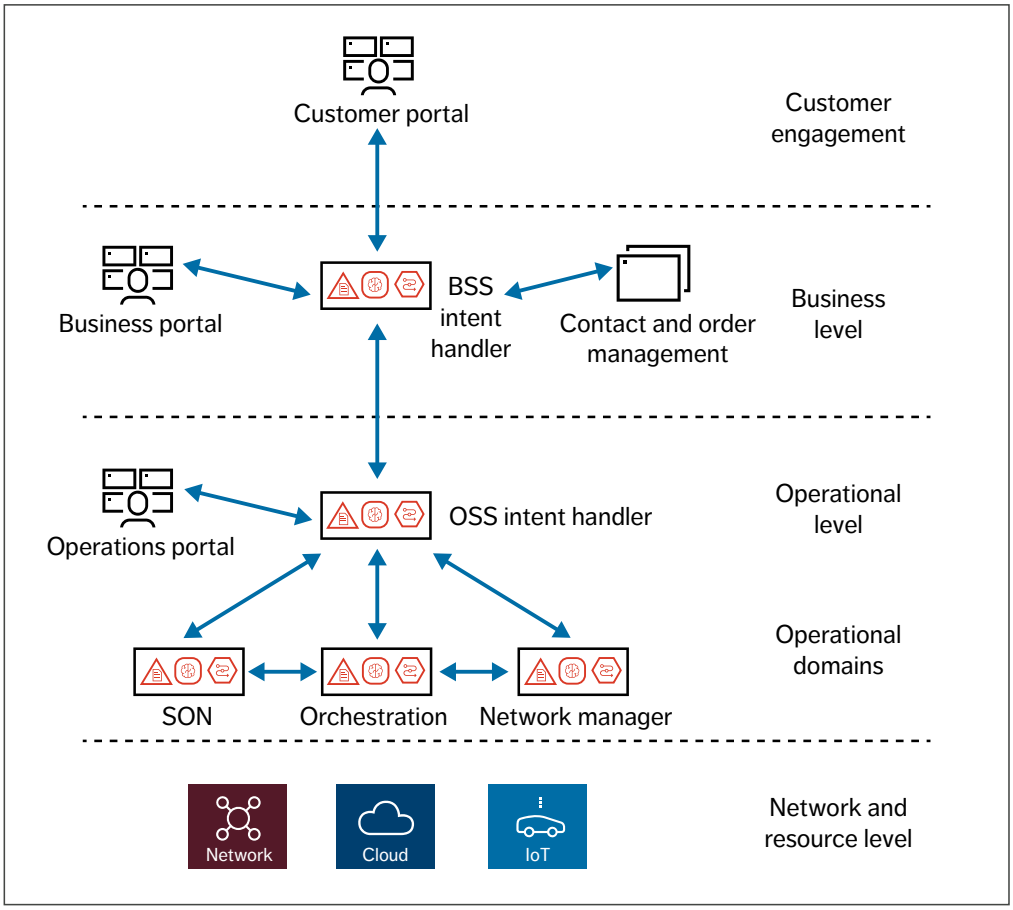


Figure 2 Intent-driven multi-layer operations system

The API (application programming interface) of the intent-handling function is domain-independent. Its main objective is to manage the life cycle of intents. It implements methods to set, modify and remove intents and send reports. Intent is constructed based on a common intent meta-model and its details are specified according to domain-specific information models. Intent management is therefore primarily knowledge management.

Figure 2 provides an example of how intent-handling functions can be combined to realize a complete intent-driven operations system. Every major system layer and subsystem domain, including BSS, OSS, orchestration and network management, contains an intent-handling function. Intent originates from functions such as contract and order management. Additional intents can be entered directly through portals.

Introducing the cognitive layer

Lexico defines cognition as: "... the mental action or process of acquiring knowledge

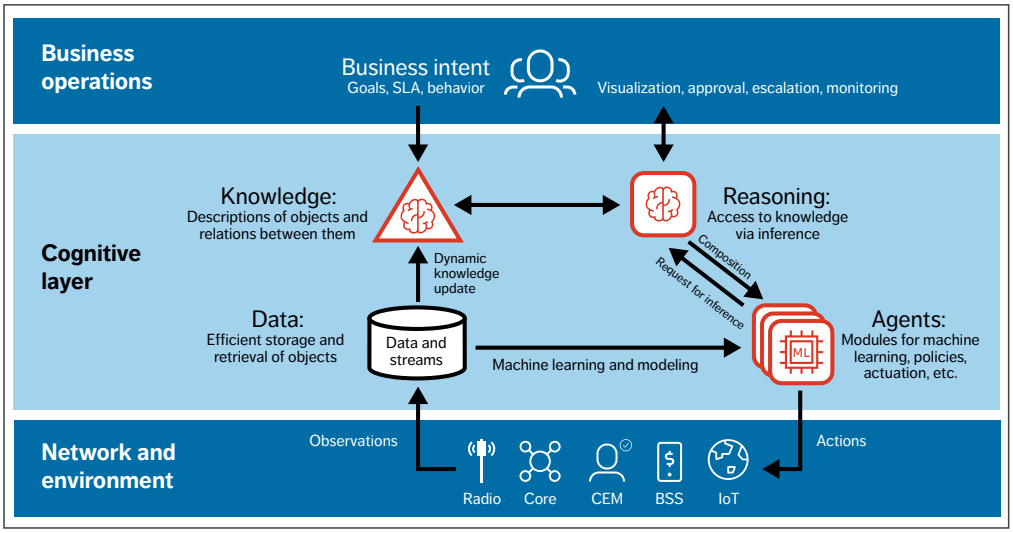


Figure 3 Functional architecture of the cognitive layer

and understanding through experience and the senses." [4] As it is designed to perform the equivalent operational tasks of understanding through experiencing and sensing, an autonomous system is, therefore, a technical implementation of cognition.

Creating an intent-handling function that understands complex and abstract intent semantics, derives the optimal target state and plans actions for transitioning the system into this state is a challenging task. The function must be able to explore options, learn from precedents and assess the feasibility of actions based on their expected consequences.

By combining well-understood AI techniques within a flexible architecture, we have designed a cognitive system that specializes in autonomous service and network operation. We refer to it as the cognitive layer, and its role is to serve as an interface between business operations and the network/environment, as shown in *Figure 3*.

The cognitive layer consists of three essential components: a knowledge base, a reasoning engine and an agent architecture. The knowledge base contains the ontology of intents along with domain-specific knowledge such as the current state of the system. The domain-independent reasoning engine uses the knowledge graph and serves as the central coordinator function for finding actions, evaluating their impact and ordering their execution. Finally, the agent architecture allows any number of models and services to be used. Agents can contain machine-learned models or rule-based policies, or implement services needed in the cognitive reasoning process.

To be usable, an agent needs to be registered and described in the knowledge base. Its description can be added and modified at any time, allowing life cycles of the models, policies and supplementary services to be decoupled from the overall life cycle of the cognitive layer.

The agent metadata contains a description of the agent interface, along with its function, role and capabilities. For example, we have implemented a machine-learned model that can propose radio base station configurations that optimize the service experience. This model is registered as an agent in the role of a "proposer" for configuration actions.

The separate life cycle makes it possible for the model to be replaced with an improved version when available, independent of the cognitive layer release cycles.

We have also demonstrated agents in the role of "predictor" with the ability to estimate the effect of actions on key performance indicators (KPIs). An agent in an "observer" role would monitor data sources, keeping knowledge about the state up-to-date. An agent in the "actuator" role can implement actions in the network by utilizing, for example, established network management functions.

> SPECIALIST AGENTS ARE USED INTENSIVELY IN EVERY STEP OF THE PROCESS

How the cognitive layer works

The successful operation of the cognitive layer depends on smooth interaction between the reasoning engine and the knowledge base. The reasoning engine continuously executes a process that tries to find actions to close the gap between the current observed state and the wanted state, according to the intent. It collects proposals, obtains predictions on the effect of each proposal, evaluates gain versus risk and certainty, prioritizes actions and executes its decisions. Specialist agents are used intensively in every step of the process.

The reasoning engine is an adaptive knowledge-driven composer that can instantiate the cognitive process following changes in intent, state and context. It can dynamically compose specialized agents and add them into the process if their capabilities and roles match the needs according to intent and context. This is, for example, how the cognitive layer obtains action proposals from agents that are implementing suitable models.

In cases where the capabilities of multiple agents match the requirements of a role, each of them can be used simultaneously to generate alternative solution strategies, resulting in a diverse set of options for the prediction and evaluation steps that follow. This coexistence of agents makes it possible to combine rule- and policy-based implementations with machine-learned alternatives in the same system, enabling the system to acquire new advanced abilities without losing current ones.

The cognitive process is a perpetual loop that starts again directly after the previous iteration has finished. Any degradation of the network or issues with services would be visible in the observed state. By trying to close the gap to the wanted state set by intent, the cognitive layer implicitly addresses incidents. Even without explicit issues, the continuous cognitive process would still seek actions for further optimization. It could, for example, try to deliver the same services with reduced resources.

Through its reasoning-based core process, the cognitive layer reaches a high degree of dynamic adaptability to new situations. This is in stark contrast to systems that have been realized through rule-based policies and fixed workflows, where every supported situation needs consideration at design time through suitable branches in the decision tree and diversifying rules. Existing rule-based policies can, however, still be used in the cognitive layer through integration as agents. This opens an upgrade path from legacy automation, with AI-based models added gradually.

Example use case

To demonstrate how the cognitive layer works, we have experimented with a use case that optimizes the provisioning decisions for the deployment of a virtual network function (VNF). The source of the intent is an SLA that specifies the service to be delivered along with the KPI targets that have been promised to the customer. Our example use case requires that the VNF be deployed with strict targets on latency and throughput.

We started by developing the required proposer and predictor agents in an offline data science process. We then deployed the service in a test environment and exposed it to a range of usage loads. We also explored deployment options by variating

parameters of the underlying TOSCA model. Combined with measured KPIs, these variations create training data sets for learning a model that is capable of connecting deployment options to expected performance. This exercise created a proposal agent that is able to recommend a TOSCA model configuration optimized for the latency and throughput figures required by the SLA.

In our experiment, online fulfillment – from receipt of the SLA to optimized deployment – is fully autonomous and controlled by the reasoning process of the cognitive layer. When the intent derived from the SLA arrives in the knowledge base, the reasoning loop starts processing it. The reasoner finds that the agent learned in the test environment matches the needs and requests a proposal. The agent delivers a TOSCA model fully configured and optimized for the requested KPI target and proposes to deploy accordingly.

The cognitive layer then uses prediction and evaluation agents to assess the proposal. In this case, our prediction agent contains a state-action model – a probabilistic graph based on Markov decision process modeling. The model is continuously learned from observing states and the results of observed actions. It has the ability to estimate the probability of expected result states for proposed actions, enabling informed decisions about actions considering their risk.

In the evaluation step, the autonomous system detects and resolves all remaining conflicts between intents. It also decides on escalations if the risk of the proposed deployment or uncertainty of the models is too high. If escalation is required, the cognitive process requests the support of a human technician, presents the situation including proposals and predictions, and asks for approval.

While the cognitive layer can operate fully autonomously, it knows when the human workforce wants to be involved. The exact threshold for escalation is at the discretion of the operator and is determined by behavioral intent. If the system gains human trust as a result of presenting many good actions, the threshold for fully autonomous decisions can be lowered.

When a proposed action is selected after verifying that the impact on all intents is advantageous and risk is reasonably low, it is then handed over for execution to the orchestrator behind the actuation agent. This concludes the first intent-handling loop after the new intent for the new SLA was introduced to the cognitive layer. The gap between the new intent and the current state was particularly wide since the new service was not yet provisioned. This gap narrowed through a provisioning action. Further iterations continuously monitor the service deployment, optimize it when possible or heal when needed. In this way, the continuous cognitive operation process provides fulfillment and assurance of expectations formulated as intents.

> “ONLINE FULFILLMENT IS FULLY AUTONOMOUS AND CONTROLLED BY THE REASONING PROCESS”

Conclusion

Zero-touch autonomy is an ideal beyond reach as long as artificial intelligence (AI) cannot match human capability to reason and decide within complex dependencies and broad domains. However, by using a combination of currently available and well-understood AI techniques within a flexible architecture, it is possible to reach a high degree of practical autonomous operation.

Our use case example demonstrates how the cognitive layer that we have developed can enable autonomous operation with the use of intents. It is built with an agent architecture that decouples the life cycles of the agents. It is coordinated by knowledge-driven reasoning that composes machine-learned models and legacy implementations. It evaluates action impact on intent fulfillment and makes decisions based on expected gains and risks, while resolving conflicting goals and maximizing utility. It can adapt to new situations through learning. The resulting system can execute many of the cognitive considerations human technicians

make when they operate services and networks manually. However, the cognitive layer does this with continuous attention and near-immediate reaction.

The adaptive reasoning capabilities of the cognitive layer enable effective management of growing network complexity, dramatically reducing the need for humans to manually modify policies or participate in online decision-making. This frees up the human workforce to concentrate on offline tasks such as executing data science processes, optimizing the available models and defining business strategies that are implemented by intent setting.

References

1. **OASIS, TOSCA Version 2.0, Committee Specification Draft 02, June 25, 2020, available at:** *http://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCA-v2.0.html*
2. **W3C, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, February 25, 2014, available at:** *https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/*
3. **W3C, RDF Schema 1.1, W3C Recommendation, February 25, 2014, available at:** *https://www.w3.org/TR/rdf-schema/*
4. **Cognition, Lexico.com, available at:** *https://www.lexico.com/definition/cognition*

Further reading

- **Ericsson Technology Review, Cognitive Technologies in Network and Business Automation, 2018, Niemöller J; Mokrushin, L, available at:** *https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/cognitive-technologies-in-network-and-business-automation*
- **Ericsson, AI by Design, available at:** *https://www.ericsson.com/en/ai-and-automation*

THE AUTHORS

Jörg Niemöller

◆ is an expert in analytics and customer experience. He joined Ericsson in 1998 and has since held multiple positions in research as well as in system management for core network and digital services. His current focus is innovation in OSS through architecture and solutions for autonomous network and service operation. Niemöller holds a Ph.D. in computer science from Tilburg University, the Netherlands, and a diploma degree in electrical engineering from the TU Dortmund University, Germany.

Leonid Mokrushin

◆ is a principal researcher at Ericsson Research. With a background in computer science and formal methods, he is currently focusing on knowledge-intensive symbolic AI systems and their practical applications in the telecom domain. Mokrushin joined Ericsson in 2007 after postgraduate studies at Uppsala University in Sweden, where he specialized in formal verification of real-time

systems. He holds an M.S. in software engineering from Peter the Great St. Petersburg Polytechnic University, Russia.

Swarup Kumar Mohalik

◆ is a principal researcher at Ericsson Research who joined the company in 2015. His expertise is in the areas of AI and formal methods, and his work primarily focuses on applying them to service automatization and the Internet of Things (IoT). He has research experience in the areas of formal specification and verification of real-time embedded software and AI planning techniques. Mohalik holds a Ph.D. in computer science from the Institute of Mathematical Sciences, Chennai, India, and a post-doctoral fellowship at LaBRI, University of Bordeaux, France.

Martha Vlachou-Konchylaki

◆ is a director of technology strategy specializing in AI and data strategy. She joined Ericsson in 2015 and has been working within different groups within Ericsson, from machine learning prototyping and business development to AI strategy. Vlachou-Konchylaki holds an M.S. in machine learning from the KTH Royal Institute of Technology in Stockholm, Sweden, and a B.Eng. in electrical and computer engineering from the University of Patras, Greece.

George Sarmonikas

◆ joined Ericsson in 2013 after several years of working for mobile operators. He currently leads AI business innovation within Digital Services. Prior to this, he was responsible for product management of Ericsson's customer experience management (CEM) and analytics portfolio, including assets for subjective experience scoring. Sarmonikas holds both an M.Sc. in communication systems and an M.Eng. in electronic engineering and computer science from the University of Bristol in the UK, as well as a graduate diploma in artificial intelligence from Stanford University, US.

ISSN 0014-0171
284 23-3350 | Uen

Ericsson
SE-164 83 Stockholm, Sweden
Phone: +46 10 719 0000