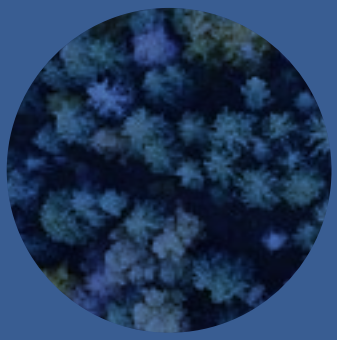
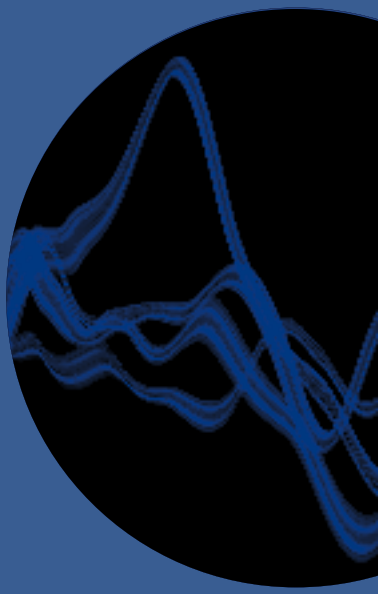
CHARTING THE FUTURE OF INNOVATION | #1·2018

ERICSSON TECHNOLOGY Review





EVOLVING OPERATIONS SUPPORT SYSTEMS

ERICSSON

OPEN, INTELLIGENT AND

model-driven: evolving OSS

The simplicity required to deploy and manage services in future network and infrastructure operations is driving the need to rethink traditional operations support systems (OSS). To unleash the potential of 5G, Network Functions Virtualization (NFV) and software-defined networking (SDN), future OSS need to be open, intelligent and able to support model-driven automation.

MALGORZATA SVENSSON, MUNISH AGARWAL, STEPHEN TERRILL, JOHAN WALLIN

The long-standing paradigm of bundling vendor and domain-specific management with network functions has been challenged in recent years by a new approach that is built on the concept of an open and model-driven platform. This new approach leverages open source and uses standardized interfaces to enable a horizontal management platform.

Today's service providers expect rapid network deployments, agile introduction of new services and cost-efficient operation and management – requirements that are even more important when planning for future 5G capabilities. Recent technology trends make it possible to redefine traditional support systems at the same time as they enable greater efficiency in development and operations.

For example, model-driven automation eliminates the need for manual interaction by externalizing behavioral aspects of specific domains from application logic. Real-time and near-real-time analytics redefine the implementation and scope of support system functions like performance, fault management, assurance and optimization, where analytics models act on the collected and correlated data, producing insights that are far richer in scope than the outcome of the traditional functions. Machine intelligence (MI) algorithms enable the transition from reactive to proactive decision making.

The DevOps paradigm simplifies development and operational processes. It requires development and deployment-friendly software architecture and

tool chains that support automation. Microservice architecture supports the DevOps paradigm and enables highly scalable, extendable and flexible systems.

Ericsson's approach to evolving OSS is based on combining these technologies and solutions to create a modern OSS architecture that is optimized to meet the requirements of service providers in the mid to long term.

Our OSS architecture principles

The key principles guiding Ericsson's OSS architecture concept are that it is service oriented, that the automation is analytics and policy driven, and that it uses virtualization and abstraction of network functions. By following these principles, we can evolve OSS to become programmable and able to leverage the interfaces offered by the production domains.

Definitions of key concepts

- **Analytics** is the discovery, interpretation and communication of meaningful patterns in data. It is the umbrella for AI, ML and MI.
- **Cloud native** is a term that describes the patterns of organizations, architectures and technologies that consistently, reliably and at scale take full advantage of the possibilities of the cloud to support cloud-oriented business models.
- **Microservice** is a small service supporting communication over network interfaces. Several such services constitute an application. Each microservice covers a limited and coherent functional scope and is independently manageable by fully automated life-cycle machinery. A microservice is small enough to be the responsibility of a single, small development team.
- **OSS business layer** is a conceptual layer of the OSS that guides the service provider through the operational support subset of its business process.
- **OSS control layer** is a layer of the OSS containing functionality that controls and manages production domains.
- **Policy** is a function that governs the choices in the behavior of a system. Policy can use a declarative or imperative approach.
- **Programmability** is the ability to externally influence the behavior of a system in a defined manner.
- **Resource** is a means to realize a service, and can be either physical or non-physical.
- **Service** is a means to deliver value to a customer or user; it should be understandable to a customer. Service is also a representation of the implementation aspects to deliver the value.

Terms and abbreviations

AI – artificial intelligence | **API** – application programming interface | **BSS** – business support systems | **ETSI** – European Telecommunications Standards Institute | **MANO** – Management and Orchestration | **MI** – machine intelligence | **ML** – machine learning | **NFV** – Network Functions Virtualization | **ONAP** – Open Network Automation Platform | **OSS** – operations support systems | **PNF** – Physical Network Function | **SDN** – software-defined networking | **SLA** – Service Level Agreement | **TOSCA** – Topology and Orchestration Specification for Cloud Applications | **UDM** – Unified Data Management | **VNF** – Virtual Network Function

> AUTOMATION CAN BE FURTHER ENHANCED BY APPLYING MI, WHICH IS THE COMBINATION OF MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

Abstraction, programmability and domain-specific extensions

Networks are complex. As the physical resources within them transform into virtual resources such as Virtual Network Functions (VNFs), virtual switches, virtual networks and virtual cloud resources, they will expose only what is necessary and be managed in a uniform way. This leads to a logical representation of resources and networks, which is referred to as abstraction. The capabilities of these virtual resources are exposed through their interfaces, while their behavior is influenced by these interfaces. This is referred to as programmability. The methodology of abstraction and programmability is continually and recursively applied at all levels of the OSS and networks, from the resources' interfaces to the exposed services.

Abstraction and programmability are the main prerequisites for uniform management of automation, easing the shift from managing networks to managing automation. They enable simplification by exposing only the required capabilities of a domain, while still supporting an efficient interaction between management system scopes, and management systems and networks.

Regardless of the generalization of the behavior between various production domains, the domain-specific aspects remain. Some examples of domain-specific extensions are: managing and optimizing radio frequency, designing and handling data center capacity, or designing L_2/L_3 service overlay. Specific production domain competencies and processes are required to provide the necessary capabilities in these examples, even though the capabilities are realized by the same OSS.

From automatic management to management of automation

The goal of automation is to provide a zero-touch network, which means that automation moves from automating via scripting what can be done manually, to an autonomic network that takes care of itself and handles previously unforeseen situations. This changes the approach to management, from managing networks manually to managing automation.

A key part of managing automation is the control loop paradigm, as shown in *Figure 1*. This is achieved by designing the insights and policies required for a use case, and the decisions that need to be made. Insights are the outcome when data is processed by analytics. Policies represent the rules governing the decisions to be made by the control loop. The decisions are actuated by COM (Control, Orchestration, Management), which interacts with production domains by requesting actions such as update, configure or heal. The control loops are implemented by multiple OSS functions and can act in a hierarchical way.

By leveraging analytics, MI and policy, control loops become adaptive and are central to assuring and optimizing the deployed services and resources.

MI and autonomic networks

Automation can be further enhanced by applying MI, which is the combination of machine learning (ML) and artificial intelligence (AI). MI adapts to the situation in a network and learns to provide the best insights for a given network situation.

As the level of MI increases, the role of policy shifts from governance of the decisions to be made, to ensuring that the insights produced from MI are appropriate. In this sense, MI is an enabler for autonomic networks.

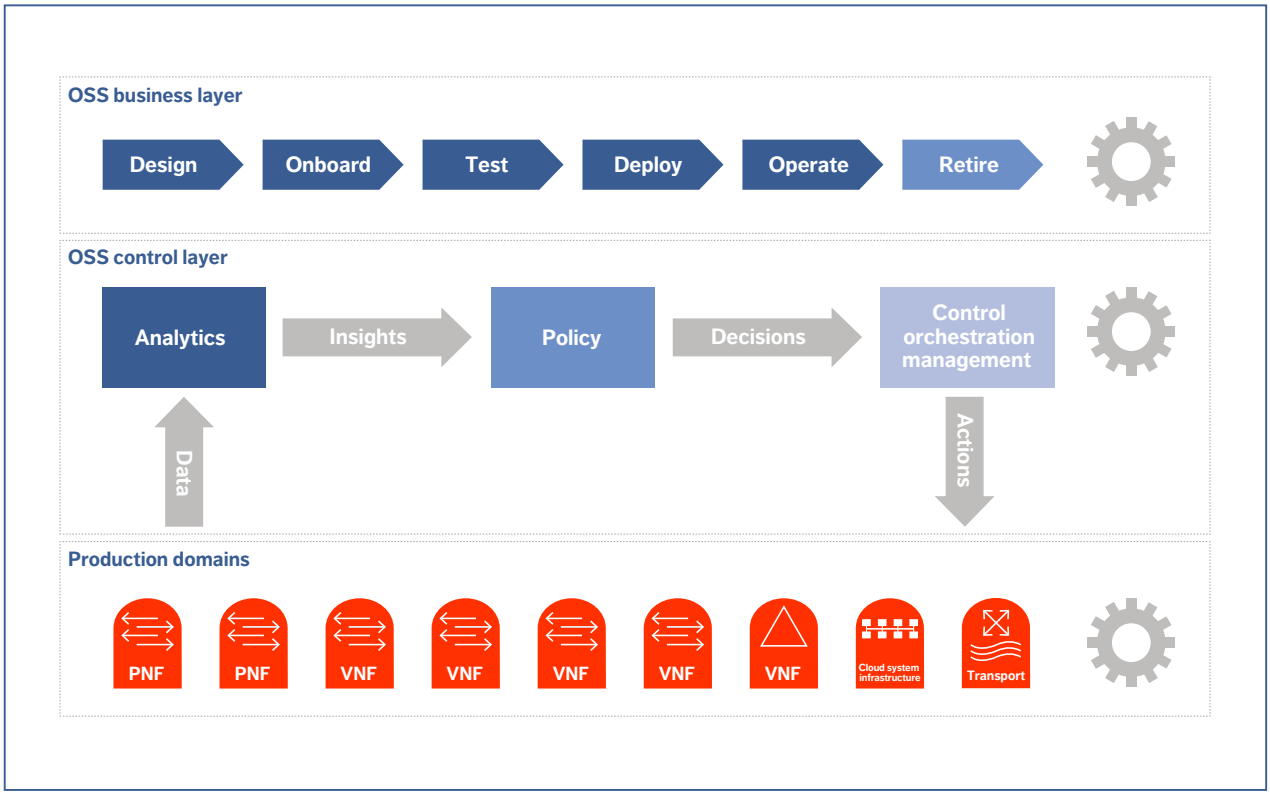


Figure 1 Analytics and policy-driven automation: closed control loop

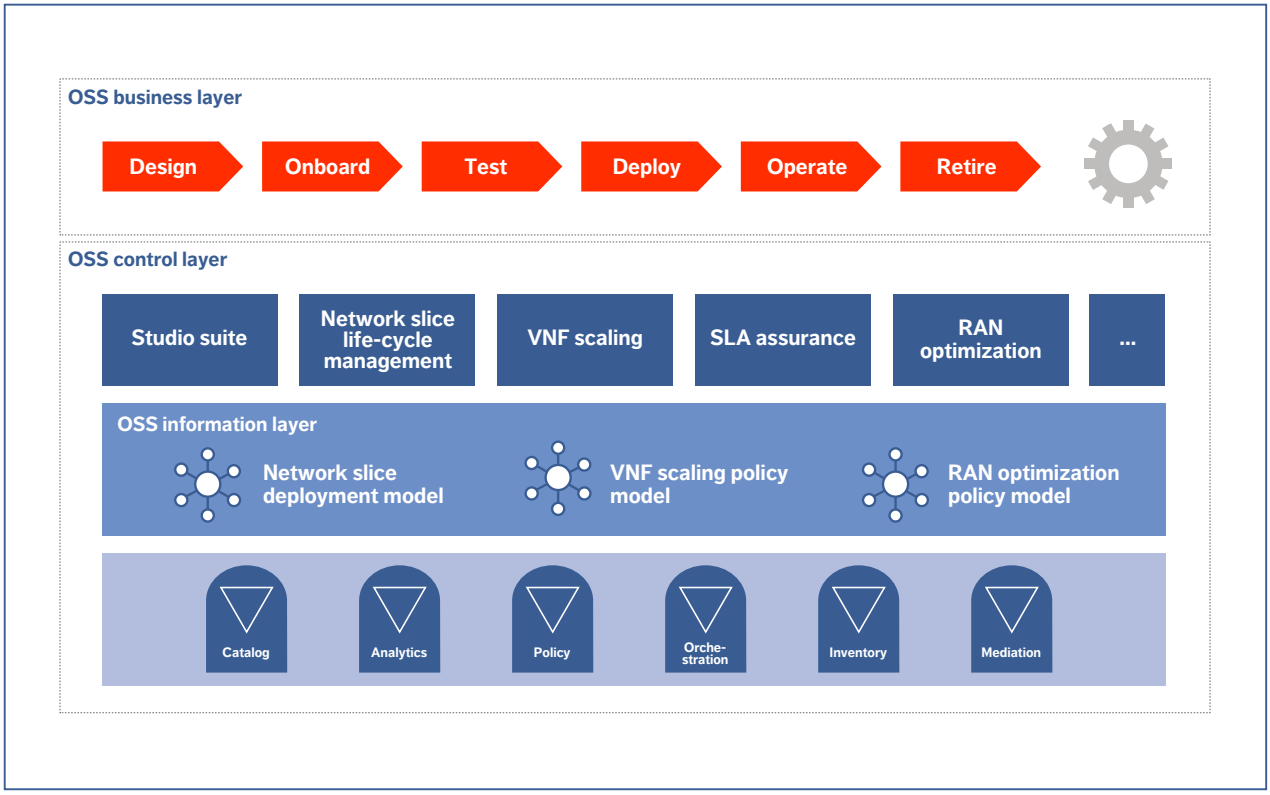


Figure 2 Model-driven OSS architecture

Model-driven architecture

To secure efficient and consistent management, the behavioral aspects of specific domains are externalized from application logic to models. There are models for configuration, orchestration, policy rules and analytics.

Resource and service life-cycle management is a good example of a model-driven approach. As shown in *Figure 2*, the models describe the services and the resources. These models are designed and onboarded in a studio suite. They are then used to test, deploy, operate and retire the services and the resources. The models are based on standard modeling approaches. For example, deployment models are described using HEAT (specifically, orchestration in OpenStack [1]) – and TOSCA [2]. The "operate" process can cover activities such as scale, upgrade, heal, relocate, stop and start.

The configuration and instrumentation of the executing resource and service instances require resource and service views expressed also as models. IETF YANG [3] has emerged as the mainstream modeling language for this purpose.

Policy rule sets define the behavioral aspects of service assurance, resource performance and scaling. Since the models are the way to capture business logic and intellectual property, they will become sellable objects themselves and be subject to commercial agreements.

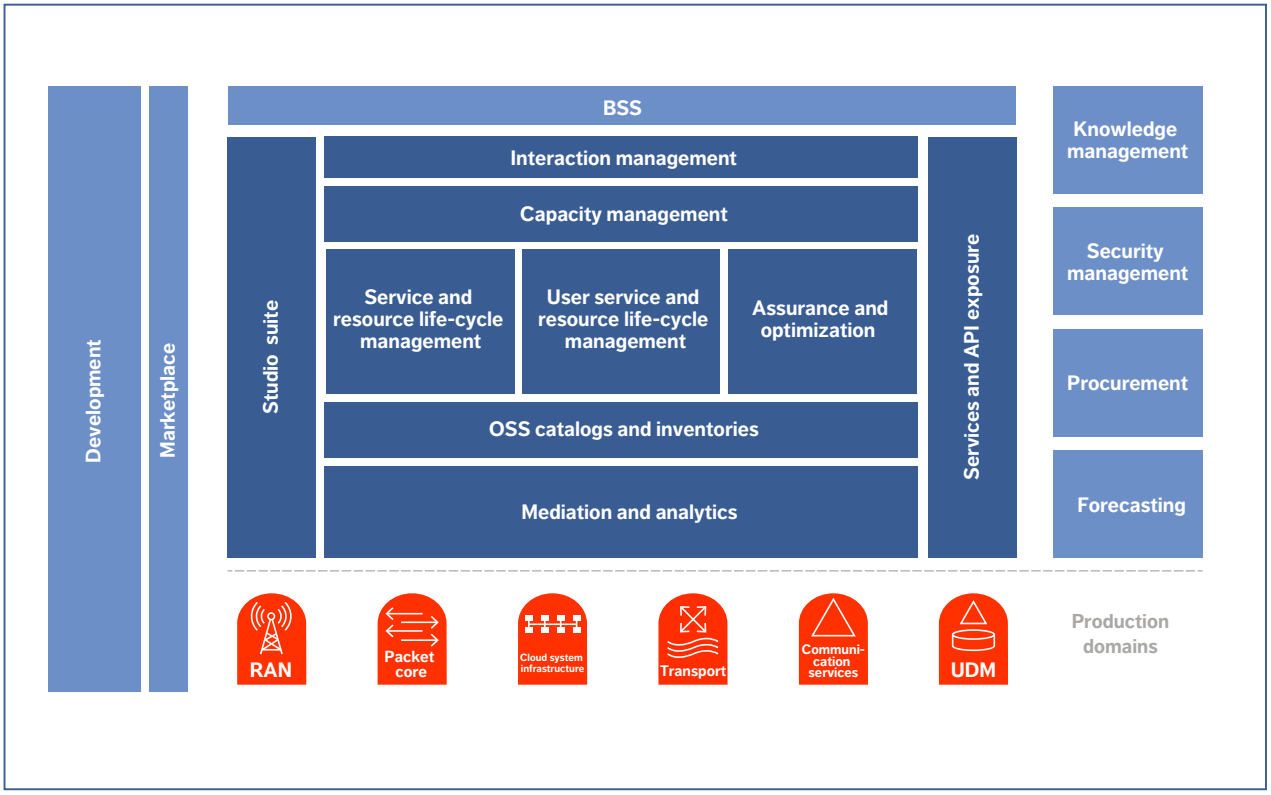


Figure 3 Ericsson's OSS architecture

Modular architecture and implementation capabilities

The optimal OSS of the future will be based both on the principles outlined above and on modular architecture and implementation capabilities.

Studio suite

Figure 3 illustrates the studio suite, which provides a single, coherent environment to design, test, operate, control and coordinate all the business, engineering and operational processes of a service provider. This is done by expressing the processes, interface extensions, policies, deployments and configurations as models and using them for automation of all the OSS functions and the managed production domains. The studio suite framework includes business, engineering and operational studios.

The business studio is the environment where business and operational processes are maintained. The business studio automatically orchestrates the process models by invoking operations on the appropriate functions in the OSS stack.

The engineering studio provides a model-driven platform where the OSS functions and the relevant information and information models associated with those functions are constructed and extended. It lets a user (human or machine) realize various management tasks by extending the OSS functions with use-case specific models for configuration,

orchestration, data and policies. It provides software development kits for defining the extension models and then instantiates the needed components, provisions the components with the models and wires them together to realize the desired use case. An example could be a set of ML functions that provide scalable machinery that can be assigned to multiple uses by applying different configuration and data models.

The operational studio offers capabilities to manage resources and services throughout their life cycle. Resources represent the infrastructure and the networks of production domains. Services use the infrastructure and networks to deliver value to customers. One of the first activities in the resource and service life cycle is to construct a specification of a resource or a service, then assign the required policy rules and configurations defining the behavior of the network, or the infrastructure or the service. Once they are defined, the specifications, policies and configurations are stored in the OSS catalogs and distributed in real time to be used by the OSS functions.

Managing services and resources

Service and resource life-cycle management functions provide a programmable way of managing the services and resources of the production domains based on specifications, policies and configurations stored in the catalogs. The core capabilities are orchestration and policy engines that parse the defined models to realize all the life-cycle steps of the services and resources. The policy and orchestration engines automate the network management by taking over the burden of low-level decision making and execution of life-cycle operations from human users.

These functions are triggered by events generated by surrounding functions like interaction management, assurance and optimization. They provide both closed loop optimization capabilities (where decision making is done based on policies and MI) and an open loop optimization feature (where a human user decides based on a set of automatic recommendations). This dynamic, model-driven and real-time interaction is what will make this future OSS solution different from the traditional one.

We can already see the need for seamless interaction between OSS software development and operations, which suggests that the core DevOps capabilities of continuous integration, delivery, release and deployment on-demand will be an integral part of the future OSS.

OSS inventories will be used to store instantiated resources and services. They track the presence, capacity and configuration of resources and services, as well as their relationships to other resources and services. They provide the data for capacity management, which enables capacity planning, design and monitoring over time. By using analytics in capacity management, it is possible to make it work in real time and be predictive and self-learning.

The purpose of the user service and resource life-cycle management function is to manage resources and services that have been assigned to a user, tenant or subscriber. This capability also supports user, tenant and subscriber provisioning. The closed and open control loop architecture patterns apply here as well. The user service and resource life-cycle management function has the same properties as the service and resource life-cycle management function.

Mediation provides an open and adaptable interface to collect data, process events generated by the production domains, and do the required translation, transformation and filtering. It also provides a set of flexible interfaces to invoke life-cycle

operations on the networks and infrastructures of the production domains.

The assurance function makes it possible to check that resources and services behave as they have been designed, published and offered. Assurance makes use of analytics to discover insights about the performance of services and resources. Policy engines are then triggered to act on the insights to determine actions that ensure the desired performance level offered on services and resources.

Our OSS architecture is designed to use interaction management, services and application programming interface (API) exposure to interact with external functions. A few examples of those external functions are development, marketplaces, business support systems, advanced security and knowledge management systems, procurement and forecasting.

The production domain example shown in Figure 3 includes a RAN, packet core and communication services, Unified Data Management, cloud system infrastructure, fixed access and transport networks. Our OSS manages multi-vendor network and infrastructure.

The OSS functions are based on an API-first approach that mandates that an API be defined before the function-exposing services can be implemented. The approach enables parallel development and provides the ability to adapt to external implementations. The API management framework facilitates the effective use of APIs during development, discovery of internal API endpoints at runtime and efficiently controlled exposure of APIs to external applications.

Cloud native

Cloud native is the new computing paradigm that is optimized for modern, distributed systems environments, capable of scaling to tens of thousands of self-healing, multi-tenant nodes. Cloud native systems are container packaged, dynamically managed and microservice oriented. The Ericsson OSS of the future will be based on cloud-native architecture to enable better reuse, simplified operations and improved efficiency, agility and maintainability.

The microservice architecture enables fine-grained reuse by having several small components instead of a single large one. By being loosely coupled and backward compatible, the services can be developed independently, enabling efficient DevOps. It also allows each service to choose the most efficient development and runtime technology for its purpose. The microservice architecture makes it possible to compose various business solutions, then deploy them automatically, significantly reducing deployment cost and time to market. The architecture will use the cloud-native ecosystem as a portability layer to support multiple deployment options.

Security

Supporting multiple deployment models requires multiple security capabilities like credential management, secure communication, encryption, authentication and authorization, which can be selected based on the deployment. Our OSS architecture will include the capabilities needed to support the multiple deployment models and to protect the models.

Embracing industry initiatives

Due to NFV and programmable networks such as SDN, there is huge industry interest in automation to decrease complexity and achieve rapid introduction of services and resources. This has resulted in several significant industry initiatives, most notably

ETSI-MANO [4] and the Open Network Automation Platform (ONAP) [5].

Initiatives that call for standardization provide an environment to develop strong concepts and identify important interfaces, while open source provides reference implementations, as well as implementation interfaces.

ONAP

By bringing together the resources of its members, ONAP has sped up the development of a globally shared architecture and implementation for network automation. ONAP provides both design time and runtime reference implementations, covering service design and creation, catalog, inventory, orchestration and control, policies and analytics. It also includes service management and automation capabilities for Physical/Virtual Network Functions (PNFs/VNFs), transport and cloud infrastructure.

Ericsson's modular and model-driven OSS approach is well aligned with ONAP and extends the ONAP vision to automation beyond network management to include interaction management, workforce management, user service and resource life-cycle management, advanced service optimization and assurance as a few examples. To interact with ONAP, we are in the process of adopting the necessary interfaces and model definitions in our product portfolio (both in OSS and network functions). Ericsson is committed to ONAP and heavily engaged with it across a broad range of topics. We aim to use the ONAP-provided open source implementations in accordance with Ericsson's architecture principles.

Managing multiple production domains requires a model-driven architecture because achieving domain-specific behavior requires the application of appropriate models on a compatible OSS platform. Since a model-driven architecture is essential to drive automation, we are working toward ensuring that our models are usable within ONAP.

Conclusion

As both an established infrastructure and OSS supplier, Ericsson has a vision for OSS in which autonomic networks enable intelligent and automated service and resource life-cycle management. We are using analytics, MI, policy and orchestration technologies to create the OSS of the future, and to offer greater efficiency in development and operations. We apply a DevOps paradigm to our software development to keep us close to customers and speed up our software delivery.

As our approach is model-driven, OSS behavior is externalized from the management platform and materialized through a combination of models and configurations. As a result, the developed models are independent of the execution platform.

Our OSS concept is modular by design and follows microservice architecture principles that make it possible to replace software components with open source implementations. The concept is built on a solid implementation architecture that enables the use of industry-defined interfaces and open source modules, as well as integration with full component compatibility.

Ericsson has embraced open source implementations through active contributions and usage, as well as driving important standardizations. We are committed to driving industry alignment that will help create a healthy ecosystem.

References

1. **OpenStack, Heat Orchestration Template (HOT) Guide, available at:** *https://docs.openstack.org/heat/ocata/template_guide/hot_guide.html*
2. **OASIS, OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC, available at:** *https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca*
3. **Internet Engineering Task Force (IETF), October 2010, YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF), available at:** *https://tools.ietf.org/html/rfc6020*
4. **ETSI, Network Functions Virtualization, available at:** *http://www.etsi.org/technologies-clusters/technologies/nfv*
5. ONAP, available at: https://www.onap.org/

Further reading

» **Ericsson Technology Review, Technology trends driving innovation – five to watch, 2017, Ekudden, E:** *https://www.ericsson.com/en/publications/ericsson-technology-review/archive/2017/technology-trends-2017*

» **Ericsson Technology Review, Generating actionable insights from customer experience awareness, September 2016, Niemöller, J; Sarmonikas, G; Washington, N:** *https://www.ericsson.com/en/publications/ericsson-technology-review/archive/2016/generating-actionable-insights-from-customer-experience-awareness*

» **Ericsson Technology Review, DevOps: fueling the evolution toward 5G networks, April 2017, Degirmenci, F; Dinsing, T; John, W; Mecklin, T; Meirosu, C; Opsenica, M :** *https://www.ericsson.com/en/publications/ericsson-technology-review/archive/2017/devops-fueling-the-evolution-toward-5g-networks*

» **Ericsson, Gearing up support systems for software-defined and virtualized networks, June 2015:** *https://www.ericsson.com/en/news/2015/6/gearing-up-support-systems-for-software-defined-and-virtualized-networks-*

» **Ericsson, Zero touch networks with cloud-optimized network applications, Darula, M; Más, I:** *https://www.ericsson.com/assets/local/narratives/networks/documents/zero-touch-networks-with-the-5g-cloud-optimized-network-applications.pdf*

» **Ericsson Technology Review, Architecture evolution for automation and network programmability, November 2014, Angelin, L; Basilier, H; Cagenius, T; Más, I; Rune, G; Varga, B; Westerberg, E:** *https://www.ericsson.com/en/publications/ericsson-technology-review/archive/2014/architecture-evolution-for-automation-and-network-programmability*

The authors would like to acknowledge the contribution their colleagues John Quilty, Bo Åström, Ignacio Más, Jan Friman, and Jaco Fourie made to the writing of this article.

THE AUTHORS

Malgorzata Svensson

◆ is an expert in OSS. She joined Ericsson in 1996 and has worked in various areas within research and

development. For the past 10 years, her work has focused on architecture evolution. She has broad experience in business process, function and information modelling; information and cloud technologies; analytics; DevOps processes; and tool chains. She holds an M.Sc. in technology from the Silesian University of Technology in Gliwice, Poland.

Munish Agarwal

◆ is a senior specialist in implementation architecture

for OSS/BSS. He has 19 years of experience in telecommunications working with product development, common platforms and architecture evolution. He has hands-on experience with mediation, rating, order management and interactive voice response. Agarwal holds a Bachelor of Technology degree from the Indian Institute of Technology Kharagpur.

Stephen Terrill

◆ has more than 20 years of experience working with telecommunications architecture, implementation and industry engagement. His work has included both architecture definition and

posts within standardization organizations such as ETSI, 3GPP, ITU-T (ITU Telecommunication Standardization Sector) and IETF (Internet Engineering Task Force). In recent years, his work has focused on the automation and evolution of OSS, and he has been engaged in open source on ONAP's Technical Steering Committee. Terrill holds an M.Sc., a B.E. (Hons.) and a B.Sc. from the University of Melbourne, Australia

Johan Wallin

◆ is an expert in network management. He joined Ericsson in 1989 and has worked in several areas in R&D in various positions covering design, product management and system

management. He has broad experience of systems architecture in various mobile telephony systems including GSM, TDMA and CDMA. For the past 10 years, he has been working with network management architectures from an overall Ericsson perspective. He holds an M.Sc. in computer engineering and computer science from the Institute of Technology at Linköping University, Sweden.

ISSN 0014-0171
284 23-3311 | Uen

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