ETSI GR ZSM 005 V1.1.1 (2020-05)

Zero-touch network and Service Management (ZSM); Means of Automation

Reference

DGR/ZSM-005ed111_Autom

Keywords

automation, management, model, network, orchestration, service

Contents

Intellectual Property Rights

Foreword

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

Modal verbs terminology

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

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

Introduction

The automation of network management and service delivery is becoming critical. All major industries are rapidly digitizing and automating their businesses, relying on state-of-the-art cloud platforms and connectivity services supporting a similar level of business agility and flexibility. CSP (Communications Service Provider) service delivery and network management automation is thus becoming critical for handling the increase in overall complexity and scale of operations created by the transformation of networks into a programmable, software-driven, service-based architecture. Going forward, unprecedented operational agility will be required to support new business opportunities enabled by technologies, such as network slicing and artificial intelligence.

The goal is to have all operational processes and tasks (e.g. service creation, fulfilment, assurance, and optimization) executed automatically and enabled at scale and at a required TCO.

The present document explores and introduces different means and approaches to automate particular aspects of these functionalities, operational processes and tasks.

Towards the automation continuum

In its simplest form, **automation** is the action of making a task executable without human intervention. It is realized by introducing new automatic functions or by replacing, modifying or augmenting manual functions with automation artifacts (e.g. a script executing a series of commands).

Automation applies to different granularities: from tasks (or function) and processes up to the entire management and operation of digital infrastructures, i.e. to the entire Life Cycle Management (LCM) of networks and services.

Communication networks and services are already well but fragmentary automated systems.

Individual functions usually exhibit high-level automation. For example, Interior Gateway Protocols (IGP) such as OSPF or IS-IS can automatically discover peers, advertise capabilities, share topology information, compute routing paths and react to failures, independently of external control or human supervision [i.1].

Automation also applies to the network or service life cycle management covering phases such as installation, configuration, provisioning, and termination; and coping with additional level of flexibility introduced by the decoupling of control and data planes and virtualisation techniques. The essential challenge arising from this advanced capability is the ability to develop integrated solutions (or systems) out of the composable components with the right levels of performance, robustness, extensibility, reconfigurability; stressing even further the need for standardized interfaces, models and mechanisms.

Yet, building a comprehensive **automation solution** remains an open problem. A comprehensive automation solution consists in chaining automated functions, with the following properties:

- Vertically end-to-end, i.e. across the protocol stack or from the service-layer to the physical-layer.
- Horizontally end-to-end, i.e. across different technologies or administrative domains.
- Repeatable and reusable in different contexts, i.e. relies on standardized or best current practices for interfaces and models.
- Provisioning dynamically, customizable "control or touch points" in the end-to-end automation loop for human supervision.

Therefore, a comprehensive automation provides an approach for combining **function automation** with **process automation**; in line with the approach of Continuous Provisioning (CP), as an additional step of the Continuous Integration (CI)/Continuous Delivery (CD) model.

Automation challenges

Realizing the automation continuum faces certain challenges.

Automation pros: machine-based automation is useful for repetitive, intensive and/or error-prone tasks. Where a human will fatigue and introduce errors, a program will run relentlessly and without departing from the normative work flow of its operations. There is no contest that in front of repetitive and intensive actions, machines outperform human capabilities by far. Automation can be applied virtually anywhere in the digital infrastructure chain of operation, from functional level to inter-functions, to processes and life-cycle.

Automation cons: Pure automation techniques show rapidly their limitations limits in front of heterogeneity, changing context/conditions of the problem at hand, i.e. they are not **adaptive**. These techniques will require:

- either advanced level of customization or fine tuning, which will make the automation solution specialized and lower the automation gain;
- or human intervention to adapt to the new context (technology, domain, operation), which works against the initial goal of automating tasks.

Tackling the automation challenge is necessary but not sufficient. Automation alone can only adapt within the function pre-defined scope and settings. Higher levels of **autonomy** can be reached by combining the automatic, **aware** and **adaptive** properties [i.2].

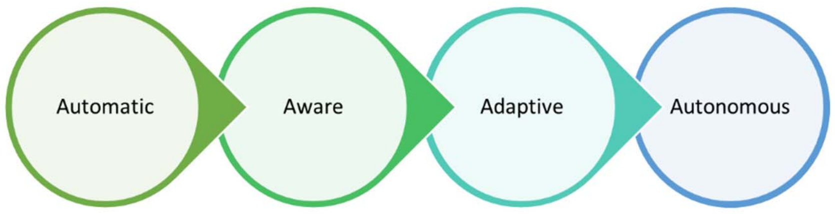

Figure 1: From Automatic to Autonomous - the "AAAA properties"

Collectively, the four properties qualify an **autonomous system**. Ultimately, this boils down to the essential coupling of automation with the **intelligence** that will drive it towards cognitive operation.

From automation to autonomous and cognitive management

Cognitive management automatically solves complex problems under uncertain (sometimes hostile) conditions to adjust and produce effective (re)action plans [i.3]. Cognitive management covers:

- Automation: ability to perform according to predetermined set of instructions.
- Complex: involve learning, inference and reasoning, causality analysis, expert knowledge.
- Uncertainty: epistemic (lack of knowledge) or aleatory (randomness/variability).
- Closed-loop: mainly adaptive (less often model-reference/predictive).

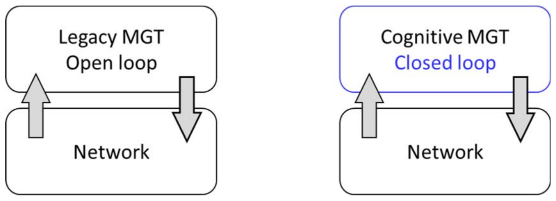

Figure 2: Legacy versus Cognitive Management - enabling a Closed Loop

Autonomous or cognitive systems relies on closed control loops for their base operation. Several types of closed control loops exist such as MAPE-K, OODA, MRACL [i.4] for the most used and well-known ones. Both individual functions and systems composed of functions can realize closed control loop operations. In the case of systems, it is the combination or chaining of the functions that collectively achieve the closed loop operation. It is therefore important to consider not only the functional-level design of the system but also the external properties of its functions or the system behaviour to assess whether the systems is autonomous (i.e. exhibits the AAAA properties).

Beyond the different types of control loops, 3 main pillars characterize cognitive system management: **measure, learn, decide.**

There is a gradient of autonomy levels for the combined management and managed systems. The share of autonomy of each part of the system can vary from case to case, from function to function and from domain to domain.

To reach a certain autonomy level, the composing parts of the system interact with each other: they discover capabilities, requirements and constraints supported by each, eventually negotiate or trade responsibilities and implications (e.g. areas of application, separation of concerns), define conditions of their interactions (escalation, delegation and coordination) to support the targeted autonomy level. This trading is dynamic. Most of the negotiation can be set at instantiation, but it should be possible to change the negotiated parameters over the life of the system. Such considerations and levels of flexibility should be considered in the design goals.

1 Scope

The present document explores different existing means or approaches to achieve automation. Such means may exist at different level of management and managed systems, e.g. at service and network management level, at managed network function level and at managed network level for autonomous optimization. All these means can have a value to achieve the ZSM goals.

The present document does not address a systematic survey of all standardization activities, open-source or other means to achieve automation. Rather it evaluates selected existing and proven means of automation at different levels of managed and management systems, provided by members of the ISG. That comprises for example:

- Alternatives for classic modelling such as intent vs. imperative vs. declarative modelling.
- Lessons learnt from 'model driven automation' of service and policy orchestration, including closed loop.
- Framework for self-managed VNFs based on cloud native network functions and implications.
- Delineate service modelling against 'machine learning'-inspired 'closed loop automation' modelling.
- Autonomous management of networks.

Means of automation may be single elements or composition of elements which enable service and network automation. They are made visible by this report, and may be considered by the ISG as elements in other work items for specifications.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

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

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

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

[i.1] OSPF for Implementing Self-Adaptive Routing in Autonomic Networks: A Case Study, Rétvári G. et al.

[i.2] Towards Autonomic Networks, S. Schmid et al.

[i.3] Interactive Self Driving Networks, P. Dimitriou.

[i.4] The Use of Control Loops in Autonomic Networking, Strassner J. et al.

[i.5] IETF RFC 7575 (June 2015): "Autonomic Networking: Definitions and Design Goals", M.Behringer et al.

NOTE: Available at https://tools.ietf.org/html/rfc7575.

[i.6] ETSI GR NFV-IFA 023: "Network Functions Virtualisation (NFV); Management and Orchestration; Report on Policy Management in MANO; Release 3".

[i.7] OPNFV Wiki, copper project.

NOTE: Available at https://wiki.opnfv.org/display/copper.

[i.8] OpenStack Wiki, congress project.

NOTE: Available at https://wiki.openstack.org/wiki/Congress.

[i.9] ONAP Wiki, Policy Framework.

NOTE: Available at https://wiki.onap.org/display/DW/The+ONAP+Policy+Framework.

[i.10] IETF RFC 3060 (February 2001): "Policy Core Information Model -- Version 1 Specification", B. Moore, E. Ellesson, J. Strassner, A. Westerinen.

NOTE: Available at https://tools.ietf.org/html/rfc3060.

[i.11] IETF RFC 3198 (November 2001): "Terminology for Policy-Based Management", A. Westerinen and al.

NOTE: Available at https://tools.ietf.org/html/rfc3198.

[i.12] Morris Sloman: "Policy driven management for distributed systems", Journal of Network and Systems Management, Plenum Press, Vol. 2, No. 4, 1994, pp. 333-360.

NOTE: Available at https://core.ac.uk/download/pdf/1587309.pdf.

[i.13] John Strassner: "Intent-based Policy management".

NOTE: Available at https://datatracker.ietf.org/meeting/95/materials/slides-95-sdnrg-1.

[i.14] A. K. Bandara, E. C. Lupu, J. Moffett and A. Russo, "A goal-based approach to policy refinement," Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004., Yorktown Heights, NY, USA, 2004, pp. 229-239.

NOTE: Available at https://www.researchgate.net/publication/4079236_A_goal-based_approach_to_policy_refinement.

[i.15] Network World Jun 2017.

NOTE: Available at https://www.networkworld.com/article/3202699/lan-wan/what-is-Intent-based-networking.html.

[i.16] Dave Lenrow: "Intent: Don't Tell Me What to Do! (Tell Me What You Want)".

NOTE: Available at https://www.sdxcentral.com/articles/contributed/network-intent-summit-perspective-david-lenrow/2015/02/.

[i.17] ONF TR-523 (October 2016): "Intent NBI - Definition and Principles".

NOTE: Available at https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-523_Intent_Definition_Principles.pdf.

[i.18] The Nemo Project.

NOTE: Available at http://nemo-project.net/.

[i.19] SDXcentral report from Cisco: "intent based networking explained".

NOTE: Available at https://www.sdxcentral.com/networking/sdn/intent-based/definitions/what-is-intent-based-networking.

[i.20] HPE service and network automation web page and white paper: "Reimagining OSS for service agility in hybrid networks".

NOTE: Available at https://h20195.www2.hpe.com/V2/GetDocument.aspx?docname=4AA6-3832ENW [i.21] Light Reading Webinar: "Making the case for NFV: it's all about the Service Model".

NOTE: Available at https://www.lightreading.com/webinar.asp?webinar_id=641.

[i.22] TMF Catalyst 2016 about policy driven: "Operations Defined Automation", based on the described intent based orchestration engine.

NOTE: Available at https://inform.tmforum.org/features-and-analysis/2016/12/will-operations-defined-automation-get-real-virtuality/?_ga=2.39738019.1872407733.1527684564-1401621168.1525266880.

[i.23] SDX central webinar, A.Krichel, R.Eberhardt, R.Chua: "Intent Based Orchestration - Enabling & Automating Next-Generation Networks", Nov 1st 2016.

NOTE: Available at https://www.hpe.com/h22228/video-gallery/us/en/700000804/EN/US/aec78df2-4a88-4fea-9b88-4b75e175755a/intent-based-orchestration-_-enabling--automating-next_generation-networks/video.

[i.24] TM Forum ZOOM project, with references to ODA and HIP.

NOTE: Available at https://www.tmforum.org/collaboration/zoom-project/.

[i.25] TM Forum HIP: "TMF070 Implementation and Deployment Blueprints for Hybrid Environments R17.5.1".

NOTE: Available at https://www.tmforum.org/resources/specification/tmf070-implementation-and-deployment-blueprints-for-hybrid-environments-r17-5-0/.

[i.26] MEF Lifecycle-Service orchestration.

NOTE: Available at https://www.mef.net/lso/lifecycle-service-orchestration.

[i.27] López, J. A. L., Muñoz, J. M. G., & Morilla, J. (2007). A Telco Approach to Autonomic Infrastructure Management. In Advanced Autonomic Networking and Communication (pp. 27 42). Birkhäuser Basel.

[i.28] EU FP7 Project EFIPSANS: "Exposing the Features in IP version Six protocols that can be exploited/extended for the purposes of designing/building Autonomic Networks and Services (2008-2011)".

NOTE: Available at https://cordis.europa.eu/project/rcn/85542/factsheet/en.

[i.29] ETSI GS AFI 002 (V1.1.1): "Autonomic network engineering for the self-managing Future Internet (AFI); Generic Autonomic Network Architecture (An Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management)".

NOTE: Available at http://www.etsi.org/deliver/etsi_gs/AFI/001_099/002/01.01.01_60/gs_afi002v010101p.pdf.

[i.30] J. Strassner, N. Agoulmine, and E. Lehtihet, "FOCALE - A novel autonomic networking architecture," in International Transactions on Systems, Science, and Applications (ITSSA) Journal, Vol. 3, No. 1, pp. 64-79, May 2007.

[i.31] Tsagkaris, K., Nguengang, G., Peloso, P., Fuentes, B., Mamatas, L., Georgoulas, S. & Smirnov, M. (2013). Unified Management Framework (UMF) Specifications - Release 3 - UniverSelf Deliverable D2.4.

NOTE: Available at http://www.univerself-project.eu/technical-reports.

[i.32] EuCNC conference, Unified Management for Converged Networks, 2013.

NOTE: Available at https://www.eucnc.eu/history/.

[i.33] M. Behringer, B. Carpenter, T. Eckert, L. Ciavaglia, J. Nobre (2016). A Reference Model for Autonomic Networking (work in progress).

NOTE: Available at https://datatracker.ietf.org/doc/draft-ietf-anima-reference-model/.

[i.34] The Linux Foundation, Open Network Automation Platform (ONAP) Project (March 2017).

NOTE: Available at https://www.onap.org/.

[i.35] M.S. Sloman: "Policy Driven Management for Distributed Systems," Journal of Network and Systems Management, Vol. 2, No. 4, pp. 333-360, Plenoum Press, December 1994.

[i.36] Davy, S., Jennings, B., & Strassner, J. (2007). The policy continuum-a formal model. In Proceedings of the Second IEEE International Workshop on Modelling Autonomic Communications Environments (pp. 65-79).

[i.37] Recommendation ITU-T M.3050.1 (2007): "Enhanced Telecom Operations Map (eTOM) - The business process framework".

[i.38] Antoniou, G., & Van Harmelen, F. (2009). Web ontology language: OWL. In Handbook on ontologies (pp. 91-110). Springer, Berlin, Heidelberg.

[i.39] Telemanagement Forum - TMF (2018). GB922 Information Framework R17.5 (SID). Parsippany, NJ.

[i.40] ETSI TS 132 500 (V11.1.0): "Universal Mobile Telecommunications System (UMTS); LTE; Telecommunication management; Self-Organizing Networks (SON); Concepts and requirements (3GPP TS 32.500 version 11.1.0 Release 11)".

[i.41] IFIP/IEEE International Symposium (2013): "Conflict free coordination of SON functions in a Unified Management Framework: Demonstration of a proof of concept prototyping platform", pp. 1092-1093, Koutsouris, N., Tsagkaris, K., Demestichas, P., Altman, Z., Combes, R., Peloso, P. & Galis, A.

[i.42] Ciavaglia, L., Ghamri-Doudane, S., Smirnov, M., Demestichas, P., Stavroulaki, V. A., Bantouna, A., & Sayrac, B. (2012). Unifying management of future networks with trust. Bell Labs Technical Journal, 17(3), 193-212.

[i.43] IEEE 7th International ICST Conference (2012): "A mathematical model for joint optimization of coverage and capacity in self-organizing network in centralized manner", pp. 622-626, Luo, W., Zeng, J., Su, X., Li, J., & Xiao, L.

[i.44] IEEE 11th International Symposium - In Modeling & Optimization in Mobile, Ad Hoc & Wireless Networks (WiOpt) (2013): "Coordination of autonomic functionalities in communications networks", pp. 364-371, Combes, R., Altman, Z., & Altman, E.

[i.45] Iacoboaiea, O. C., Sayrac, B., Ben Jemaa, S., & Bianchi, P. (2014). SON conflict resolution using reinforcement learning with state aggregation. In Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges (pp. 15-20). ACM.

[i.46] H. Lipson: "Robots on the run," Nature, 28 March 2019.

NOTE: Available: https://www.nature.com/articles/d41586-019-00999-w, accessed 1 August 2019.

[i.47] K. Goldberg: "Robots and the return to collaborative intelligence" Nature Machine Intelligence, vol. 1, pp. 2-4, 2019.

[i.48] X. Zhao, L. Xia, T. Jiliang and Y. Dawei: "Deep Reinforcement Learning for Search, Recommendation, and Online Advertising: A Survey," ACM SIGWEB Newsletter, 2019.

[i.49] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu and Veness, "Human-level control through deep reinforcement learning," Nature, vol. 518, pp. 529-533, 2015.

[i.50] D. Silver, A. Huang, C. J. Maddison, A. Guez and L. Sifre: "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, pp. 484-489, 2016.

[i.51] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup and D. Meger: "Deep reinforcement learning that matters," in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[i.52] IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (2018): "Training Deep Networks With Synthetic Data: Bridging the Reality Gap by Domain Randomization", J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon and S. Birchfield.

[i.53] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[i.54] "Machine intelligence addresses increasing network complexity", 27 February 2018.

NOTE: Available at https://www.ericsson.com/en/press-releases/2018/2/machine-intelligence-addresses-increasing-network-complexity.

[i.55] V. Yajnanarayana, H. Rydén, L. Hévizi, A. Jauhari and M. Cirkic: "5G handover using Reinforcement Learning", arXiv:1904.02572, 2019.

[i.56] Bao, Y., Li, Y., Huang, S., Zhang, L., Zamir, A. R., & Guibas, L. (2019): "An Information-Theoretic Metric of Transferability for Task Transfer Learning. ICIP".

[i.57] Berger, S., Simsek, M., & et al. (2016): "Joint downlink and uplink tilt-based self-organization of coverage and capacity under sparse system knowledge". IEEE Trans. on Vehicular Technology, 2259-2273.

[i.58] Engels, A., Reyer, M., & et al. (2013): "Automonous self-optimization of coverage and capacity in LTE cellular networks". IEEE Trans. on Vehicular Technology, 1989-2004.

[i.59] Gumbira, A. (2018): "Transfer Learning in Deep Convolutional Neural Networks". Dissertation Instytut Informatyki.

[i.60] Mnih, V., Kavukcuoglu, K., & et al. (2015): "Human-level control through deep reinforcement learning". Nature, 529-533.

[i.61] Pan, S. J., & Yang, Q. (2010): "A survey on transfer learning. IEEE trans. on knowledge and data engineering", 1345 - 1359.

[i.62] Parera, C., Redondi, A. E., Cesana, M., Liao, Q., & al., e. (2018): "Transferring knowledge for tilt-dependent radio map prediction". IEEE Wireless Communications and Networking Conference (WCNC).

[i.63] Parera, C., Redondi, A. E., Cesana, M., Liao, Q., & Malanchini, I. (2019): "Transfer Learning for Channel Quality Prediction". IEEE International Symposium on Measurements & Networking (M&N).

[i.64] Xu, X., Sun, Z., Dai, X., & et al. (2017): "Modeling and analyzing the cross-tier handover in heterogenous networks". IEEE Trans. on Wireless Commun., 7859-7869.

[i.65] ETSI GS ZSM 001: "Zero-touch Network and Service Management (ZSM); Requirements based on documented scenarios".

[i.66] ETSI GS ZSM 007: "Zero-touch network and Service Management (ZSM); Terminology for concepts in ZSM".

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

Void.

4 Means of Automation

4.1 Overview

As mentioned in the Introduction, the present document explores and introduces different means and approaches to automate particular aspects of functionality, operational processes and tasks of service and network management. The goal is to achieve higher level of the automation continuum, up to **autonomous** systems, combining the **automatic**, **aware** and **adaptive** properties, finally enabling a **cognitive** management with a closed loop.

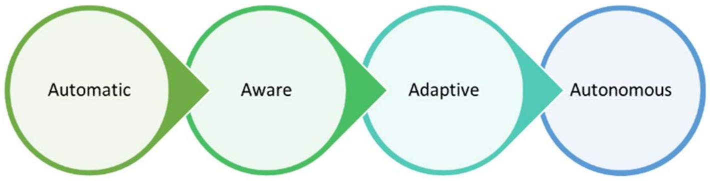

Figure 3: From Automatic to Autonomous (see Figure 1 in Introduction)

The subsequent clause provides a selection of such approaches, documented by the ZSM ISG as valuable and proven examples of means for automation, when it comes to an implementation of zero-touch management systems.

Each means for automation clause follows a common structure comprising:

- The business motivation.
- The technical problem statement.
- The means' concepts and principles and its implications to a solution.
- The elements used to demonstrate the evaluation of the means for automation are explained as "Proof of concept" may vary and could range from a theoretical evaluation study up to concrete experiences of vendors or operators with evidence of installations in operational environments.
- A final clause "Relevance for ZSM" will address the evaluation of the means for automation and relevance for the ZSM ISG.

The contributions provided to the present document may be structured into three areas:

A. Policy-based operations automation.

B. Advanced challenges and concepts managing autonomic functions.

C. Machine learning.

A. Policy-based operations automation

Looking at implementation and concrete means of operations automation, policies are obviously a most important element to automation functions. Clause 4.2 outlines the most typical use of policies.

The more sophisticated evolution of policies is outlined in clause 4.3. It differentiates imperative from declarative policies, and introduces the notion "Intent" as a further simplification of declarative polices. While several solutions for zero-touch management systems on the market today claim the characteristics of "intent-based", the differentiation often becomes difficult. The clause provides some history of the notion, and provides a differentiation on various levels of a zero-touch management solution, such as API level, network controller level, Network service orchestration level, and higher level service orchestration.

A more holistic, proven and concrete market example of the higher level service orchestration is provided with clause 4.4. It shows how an evolved service description language allows a generic orchestration engine, to evaluate necessary orchestration actions at runtime. The concept allows the drastic improvement of time-to-market for new services.

B. Advanced challenges and concepts managing Autonomic Functions (AF)

When translating business goal (e.g. an "Intent") into concrete management of autonomic functions (as defined in IETF RFC 7575 [i.5]), a framework is necessary. Clause 4.5 explains the advanced concepts for Network and Service Governance. It structures the technical challenges in five main functional areas: business language, translation, reasoning, policies, and configuration enforcement.

Specific challenges appear, when a number of AFs compete with one another. To achieve stability such conflicts have to be detected by monitoring their behaviour first, second it requires a role of coordination to adapt their behaviour. Clause 4.6 provides the approach to address such coordination.

C. Machine Learning

Machine Learning is an essential means for zero-touch automation: algorithms can learn from collected information, identify patterns and generate models for various optimization and classification tasks. Among the multiple techniques of ML, two of them are described in more detail hereafter, as they address the specific challenges in highly dynamic environments:

- Reinforcement Learning (RL as in clause 4.7 is able to learn policies using an agent in highly dynamic environments, where predefined models may not work. It can be seen as a way of creating a solver to a problem by error-and-trial. Deep Reinforcement Learning (DRL) is a most successful way of solving the complexity of RL in hard scenarios. It uses basically the state-of-the-art of neural-networks.
- Transfer Learning (TL) as in clause 4.8 is a technique to transfer such knowledge from one agent to another. It tries to avoid long learning cycles, but can be also well combined, e.g. with Deep Reinforcement Learning (DRL).

4.2 Means of Automation: Policy Driven Automation

4.2.1 Motivation

In many cases, automatic actions can be described using policies. Many network deployments already provide dynamic policy management that can drive dynamic changes in life cycle management and other network configurations.

See also the scenario on policy driven automation, clause 6.2.3.8 in ETSI GS ZSM 001 [i.65].

A large number of scientific documents and other standards for policy management are available. As a starting point see IETF RFC 3060 [i.10], IETF RFC 3198 [i.11] and [i.12].

Using policy based management, the trigger to execute measures of automation is moved more closely to the network, improving the speed of reaction and decoupling the OSS/BSS decisions from the fast execution in the managed entities.

In most cases, policy driven automation will be used in combination with other means described in this clause.

4.2.2 Problems to be solved

The key idea with policies is to create pre-defined rules that trigger automatic actions when certain situations occur. Possible situations include:

- Time of day.
- Load thresholds.
- Failures.
- Combinations of above.

In case the trigger situation occurs, the local management entity would execute pre-defined actions and notify ZSM afterwards. The predefined actions include:

- All lifecycle operations:
  - Instantiation of services or components.
  - Termination of services or components.
  - Scaling.
  - Migration.
  - Replacement of instances.
  - Configuration changes.
  - Software management.
  - Other.

In order to achieve policy driven automation, a number of issues need to be solved:

- Processes of policy definition.
- Language for the policy definition.
- Interworking of different policy definition languages.
- Management of policies (policy repository).
- Policy enforcement (detection of conditions, policy decision, policy execution).
- Security issues at the time of creation and management of the policies.
- Security issues at the time of execution.
- Mapping to administrative roles and domains.
- Detection and resolution of conflicts between policies.

The above is deliberately focusing on the technical side of policy driven automation. However there are other aspects to be considered such as trade-offs between network operations and business decisions.

4.2.3 Solution Principles and Concepts

Policy driven automation can best be achieved if policy management is part of the managed entity (see "managed entity" in ETSI GS ZSM 007 [i.66]), i.e. the deployed elements in the network managed by the ZSM framework.

In these cases the managed entities will support their capabilities for policy based automation:

- Dynamic policies can be defined for the network services or network functions.
- These policies define automatic actions of life cycle management, which will be executed by the policy execution function in the orchestrator of the deployed system.
- ZSM framework can create and activate the policy definitions for each part of the deployment it manages.
- Policy definitions can be made very specifically, e.g. for each instance of a managed entity or on domain level.

In case some managed entities do not provide policy management or cannot control certain actions by the policies, the ZSM framework can provide its own policy based management. This still provides a capability of programmable automation, even when the managed entities cannot act autonomously (see option 2 in clause 5.1.3 of ETSI GS ZSM 001 [i.65]).

In both cases ZSM framework can provide capabilities to dynamically change policies and by that allow users to adapt the dynamic behaviour of the automation to his needs.

The ZSM framework needs to make sure policies are consistent over the whole system it manages. Conflicts of policies are a well-known challenge to any policy management. ZSM framework can address these issues only if it is established as a master over all policies in the systems it manages. Necessary means for the interaction between multiple originators of policies need to be analysed and established. (see also clause 4.6).

4.2.4 Implications

Policy driven automation allows full zero-touch management as soon as the policies are in place. However creation of the policies and dynamic adaptation of policies to changed situation is a challenging task. There are different approaches to apply and manage policies, which may have different Implications. The use of AI and ML techniques can support such tasks, also providing mechanisms to translate a business intent into specific policies.

See also clauses 4.3 and 4.4 as example reports about intent-based.

4.2.5 Proof of concept

Many initiatives include concepts, descriptions or implementations of policies. As an example see from open source the copper projects in OPNFV [i.7], which integrates solutions from OpenStack (Congress [i.8]) and OpenDaylight (Group Based Policy [i.9]).

A description of a policy framework can also be found in ONAP [i.9]

See a description of an implementation independent policy framework in ETSI GR NFV-IFA 023 [i.6].

A PoC project under the ZSM PoC framework would be beneficial for the industry.

4.2.6 Relevance for ZSM

As shown above, ZSM framework should implement policy driven automation including following functionality to achieve zero-touch automation:

- Make use of policy management provided by managed entities.
- Translate service provider intent into policy definitions.
- Implement policy management and enforcement for programmable automation of managed entities that do not provide policy management on network service layer.
- ZSM framework needs to provide the overall policy management including lifecycle management for the policy definitions.

Depending on the complexity of the policies, more sophisticated automatic management can be achieved than with simple closed-loop automation.

ISG ZSM should analyse how to achieve a solution independent framework, e.g. common policy definition language, policy-related APIs, interactions for resolution of conflicts.

4.3 Means of Automation: "Intent Based"

4.3.0 Introduction

The notion "Intent" appeared in the Telecom Industry around 15 years ago. Actually, its origins are rooted in policy-based management much earlier, described by M.S. Sloman in 1994 [i.12]: "*Policies are rules governing the choices in behaviour of a system*". John Strassner refined it in 2003 [i.13]: "*Policy is a set of rules that are used to manage and control the changing and/or maintaining of the state of one or more managed objects*". Strassner differentiated imperative from declarative policies, and looked at various approaches to manage policies (e.g. DEN-ng, IETF SUPA) and concluded, policies need to be more simple. Strassner discussed "*Intent" as a more simple approach than declarative policies. In his "policy continuum*" (see [i.13]) he identified the layers of business view and system view, for which Intents are applied.

Around 2004 a lot of studies around policy refinement and conflict management were done. A most mentionable study by Arusha Bandara, Emil Constantin Lupu et al (see [i.14]) describes "the Goal-based Approach to Policy Refinement". It describes how Policy-based management uses rules to govern behavioural choices, whilst satisfying the goals of the system. One can say, a Goal is a synonym for Intent.

Finally, in 2015 Dave Lenrow [i.16] described the fundamental principles of what Intent Based means. In this paradigm, intelligent software understands the user goals, and translates them automatically into a concrete prescription of service or network configuration.

As such, the Intent-based approach becomes a very concrete means of automation. However, the notion has various aspects, is interpreted in different variants, and there is no agreed standardized definition today. Indeed not all approaches called "intent-based" have a direct meaning for automation; some just refer to the model of the northbound API what allows to ease integrations.

This clause explains and refers to some essential concepts and principles, and differentiate various interpretations of "Intent-based". That has implications to architecture and concrete software; more specific it indicates what aspects can become meaningfull for ZSM.

4.3.1 Motivation

The first motivation for Intent-based networking is the increasing and inherent **Complexity and Dynamics** of networks, based on NFV and SDN, where network functions are dynamically distributed and scaled across many places. The distribution changes dynamically, e.g. moving across different locations, edge, cloud, central, following the usage need, for example with 5G when requesting individual slicing for specific application need. Classic approaches with separated fulfilment and assurance fail, as the impact detection and root cause analysis are not able to resolve the continuously adapting network configuration.

The second motivation is **Simplicity.** For example enabling an intuitive self-service without any technical details, but also enabling a faster way of service composition to build value-add products using existing services, at immediate effect. Most flexible and simple composability of services is most important to gain new revenues with shortest time-to-market.

That motivates the third topic: **Agility**. Accelerate speed and hence reduce cost of service development including a zero-touch management. Intent-based principles allow the definition of a service model, which accelerates the DevOps process massively, enabling a CI/CD approach.

A final motivation is more **Business** serving different industries with digital services based on programmable infrastructures, enabling new digital ecosystems. Communication service providers may offer applications on top of quality connectivity services, enriched with SLAs and security services. A broadcast company, a car manufacturer, or a hospital, may order their applications with specific configurations and quality, directly connected from a private cloud running on an edge, with specific slice connectivity. All requires orchestration of complex service compositions.

4.3.2 Problems to be solved

Complexity of networks and services grows, due to increasing number of network technologies, network layers, network dynamics, essentially driven by virtualisation. The request for fulfilment of a connectivity can become very complex and may require many different variants depending on multiple factors and parameters. The number of necessary change workflows grows exponentiallly.

On the assurance side the detection of a real service impact, based on faulty resources becomes impossible as network topologies getting too dynamic. It requires new approaches to compose network service across multiple domains and layers with different technologies. Assurance need to become agnostic to the dynamics of network changes, or assurance becomes an implicit part of the single managed service or resource itself.

Intent is a way to abstract complexity at an interface level. However, the translation of the intent into a concrete network configuration requires data and context for a policy-based dynamic instantiation with late bindings at runtime. A classic workflow approach, matching to a specific hierarchy (defined as template) may be too static. The bigger the hierarchy to manage, the more complex and hard it is to adapt.

"Zero touch" requires autonomy of systems. Composition of smaller autonomous service objects allows more agility. Nevertheless, how can a single service object achieve more autonomy, while mitigating the risk to break the whole system?

Composition is easy in a model-driven approach. In typical management systems today, often code needs to be adapted, when the service model changes. A truly model-driven automation requires a modelling language supporting appropriate semantics and avoids coding. Classic modelling techniques typically document a desired service hierarchy, but cannot drive the behaviour of a generic engine; they need additional interpretation. Multi-level graph modeling is a precondition for flexible service composition, inheritance needs to be supported e.g. to enable a late binding at runtime, in particular it needs semantics to express behavior with policies. Furthermore, the language needs support of a state model to associate state transition actions, and relationships need to support dependency calculations in some way.

4.3.3 Principles and Concepts

4.3.3.0 Differentiating Intent from Policy

Before the notion Intent-based was defined, there was a Policy-based management. An Intent-based management may be implemented using classic means of separating service-models, policy-management and workflows, but an Intent-based modelling allows a fully model-driven automation.

4.3.3.1 Policy-based management

John Strassner differentiates policies (see [i.13]):

Imperative policies define a triple: event, condition, action ("ECA-policies"). Depending on a current state, the *condition defines, what action* to take to achieve a desired state. The **policy author defines what to do** under what condition.

Declarative policies, also known as "goal-policies", specify the *desired state*. They let the **system compute the action**.

The essential difference is: declare WHAT is wanted, not HOW to do it. It avoids coding, and allows even to deal with unexpected situations.

There can be policy conflicts in both approaches, therefore Strassner defined a third policy type:

Utility function policies allow the **system to compute an optimal action**, based on an objective based on the resulting state, such as "maximize" or "minimize".

However, Strassner concluded, policy management is hard and people want simpler solutions, and motivates the need to introduce "Intent". Approaches to formalize policy descriptions (such as TMF's DEN-ng, IETF SUPA, etc.) became complex to manage. In particular product managers and end users are not technical. Service designers want to build Network Services, but various network interfaces do not help on this.

Comparing to above, an Intent is a simplified goal-policy.

4.3.3.2 Intent-based management

As mentioned above, an Intent is a simplified declarative policy.

An Intent-based API allows the request for an Intent.

An Intent-based networking system will use different policies (as above) to translate the intent into a concrete network setup.

Strassner defined the policy continuum (see [i.13]), while he saw in the intent only at the business and service level. Today an Intent may apply even at Instance level, given the specific resource or network manager supports an Intent API.

While policies are concrete functions to take decisions, Intent has higher-level goals and benefits. Dave Lenrow's definition in 2015 [i.16] was made in context of Intent-based networking. SDN was the primary focus that time. In the meantime network technologies are getting more hybrid, SDWAN takes NFV into account, NFV uses SDN technologies etc. Some of Lenrow's principles may experience a wider scope in the meantime, while all its fundamental benefits are still valid:

Intent is invariant and portable - An intent may be changed by the user over time, but an intent never changes as a result of a fault or change in the infrastructure. The invariant description of an intent frees management applications from the underlying network details, simplifying overall application and service development, testing, and deployment. An Intent expression is technology agnostic by design. As a consequence an Intent is portable. E.g. Intent-based networking eliminates the impact of network changes. Intent allows portability across a range of dissimilar solutions including the SDN controllers, eliminating lengthy applications integration changes and run-time complexity as a result of inevitable changes to the infrastructure.

Invariance and portability match essentially the concept of the "CFS" in TMF SID (see [i.39]), which offers abstraction of any network or resource specifics (of course, also "RFS" do abstract resource specifics).

Intent is composable - An extensible intent API is designed to allow disparate services, developed independently, to express their requirements in a simplified language.

Today's services may be based on heterogeneous resource management systems (private or public cloud, edge, central, SDN, SDWAN, NFV, MPLS, etc.). An Intent for "Connectivity" may be assembled using various resource management systems. Therefore a user's intent may be composed out of several resource intents, finally executed by different resource (or domain) managers.

As Intent can be composed from other Intent, they can be interacting within or across management systems. There is no limit of intent levels, nor a fixed hierarchy. Intent models can just the scope of a single orchestrator or across e2e and several domains.

Intent-based APIs just declare the offered Intent of a management system.

Intent scales - Given on context, intent can cause scaling (out/in/up/down) as needed. Taking technology agnostics into account, a system may decide autonomously how it scales to achieve the desired capacity.

Intent provides context and dissolves conflict - as example, when different SDN services push low-level (e.g. OpenFlow) rules, there is always a risk of conflicting changes to the system state. Attempts to examine these rules (of the form "match this header and perform this action") and resolve such issues have been unsuccessful because at this low level of abstraction, it is impossible to decode the overall intent of the services pushing the rules. Intent-based modelling languages (see next clause) provide an intent-oriented description that conveys the "why", rather than the "how". Hence, it is possible to determine actual or apparent conflicts and seek ways to fulfil the cumulative intent of the multiple-client services. Beyond SDN this approach applies to any other service technology managed thru intent.

4.3.3.3 Intent-based modelling

While all above definitions may describe policy description and simplified interaction with or across management systems, the way to translate an intent to a prescription by a system is not defined yet. This is also where vendors solutions and interpretations differ.

Given a Goal is an Intent, a basis for Intent-based modelling was laid down in 2004 with the "goal-based approach to policy refinement" by Lupu et al [i.14]. It describes a graph of objects, representing goals, where each gaol can compose sub-goals, either conjunctive or disjunctive, depending on operations associated with the goal. The approach used *Event Calculus* as a formal language for representing and reasoning dynamic systems.

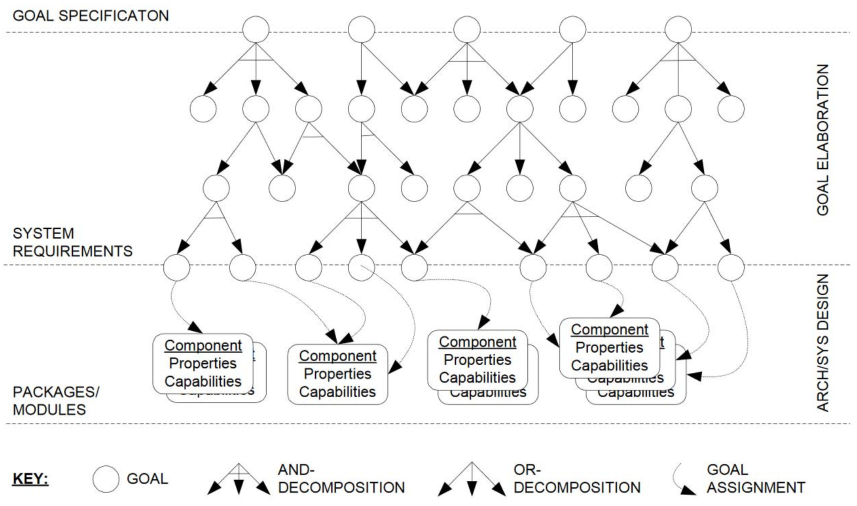

Figure 4: Goal based policy refinement, source [i.14]

Intent-based modelling can be used as a new name for a goal based policy refinement. At the highest level, a CFS with a certain quality can be seen as a user goal. But the essential difference to classic service modelling hierarchies (CFS-RFS-Resource), is the combination of the intent-graph with its policy evaluation (behaviour) for composition (instantiation of relationships) at runtime. These policies can be more complex than an AND/OR decomposition. Classic descriptor languages have no elements to define the behaviour within the descriptors. Intent-based modelling languages need elements to support automation.

For the intelligent engine, the notions CFS or RFS or Resource are less relevant; it just follows the atomic objects' goal-policies. In addition, the service objects can be reused for different purpose: am RFS today (e.g. a Firewall as part of a Security CFS), can be a CFS tomorrow (when the customer wants to order a dedicated Firewall to build his own Services). Just a point of view and of intent abstraction. The underlying model is a graph, rather than a hierarchy. Just the label CFS makes the intent available to the NBI.

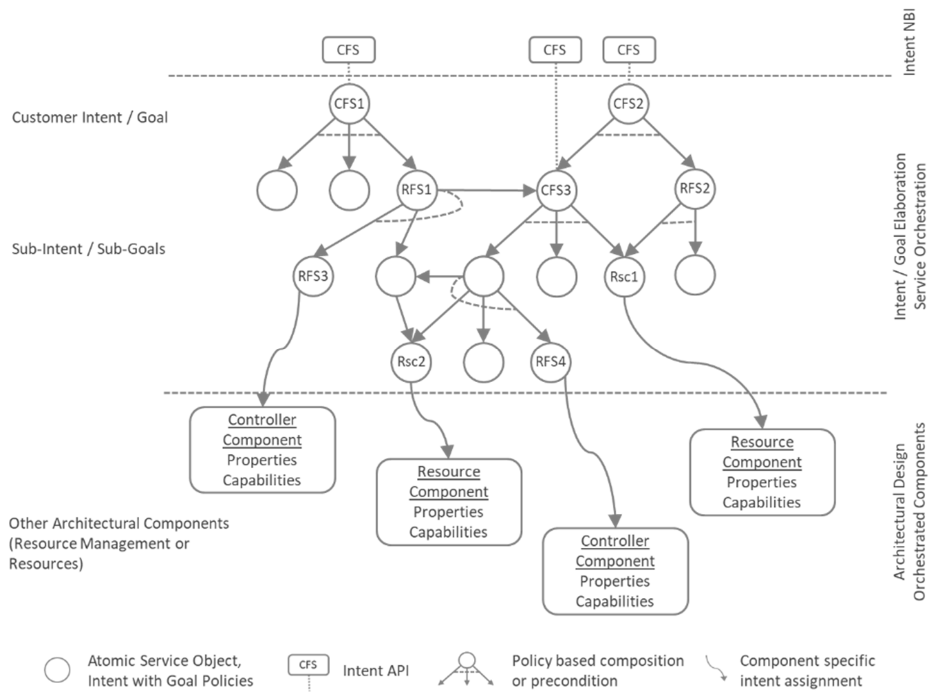
Intent NBI
CFS
CFS
CFS
Customer Intent / Goal
CFS1
CFS2
RFS1
CFS3
RFS2
Sub-Intent / Sub-Goals
RFS3
Rsc1
Rsc2
RFS4
Intent / Goal Elaboration Service Orchestration
Controller Component Properties Capabilities
Resource Component Properties Capabilities
Resource Component Properties Capabilities
Controller Component Properties Capabilities
Other Architectural Components (Resource Management or Resources)
Architectural Design Orchestrated Components
Atomic Service Object, Intent with Goal Policies
Intent API
Policy based composition or precondition
Component specific intent assignment

Figure 5: Intent-based policy evaluation graph

In the meantime, there are a few vendors on the market with different approaches to formalize Intent-based modelling. There is no agreed standardized concept of Intent-based modelling, driving a generic engine, agnostic of any specific domain.

Intent-based modelling does not necessarily match with familiar models, such as typical service hierarchies. It is a graph, which allows the engine to decide any time, what dependencies are necessary to instantiate. The single service objects are reusable (composable), together with their policies. This differentiates from template approaches, where one workflow covers one defined hierarchy. An Intent-based modelling is rather intuitive from a procedural point of view (to achieve the goal x, I need to achieve goal y, with a given context of z).

A typical problem solved with the approach is the use of shared resources. In Figure 5 for example, "Rsc2" is used as part of "RFS1" as well as part of "CFS3". Thanks to the modelling, the engine automatically recognizes, when "Rsc2" can be deleted; it releases the resource, exactly when all requesting relationships are gone, without any additional management overhead.

Architecture components being integrated can be anything, resources, resource managements, subordinated service management, etc. As there is no mandatory need for hierarchies anymore, architecture components can be associated with any service object in Figure 5.

4.3.4 Implications

4.3.4.0 Notions of "Intent-based"

The following notions of Intent-based may apply to an architecture:

- Intent API.
- Intent-based networking systems (IBNS).
- Intent-based network service orchestration (IBNSO).
- Intent-based service orchestration (IBSO).

4.3.4.1 Intent API

Intent-based API is primarily the abstracted and simplified API to request an intent. Instead of a descriptive call, for example how exactly a router is configured, just a prescription is given, that I need a connectivity between some points. That comprises invariance and portability.

ONF described the Intent API in [i.17] as a paradigm, which allows using simplified prescriptive *keys* at API level, which are converted into descriptive values for a specific domain controller by an Intent engine, using key-value mapping tables.

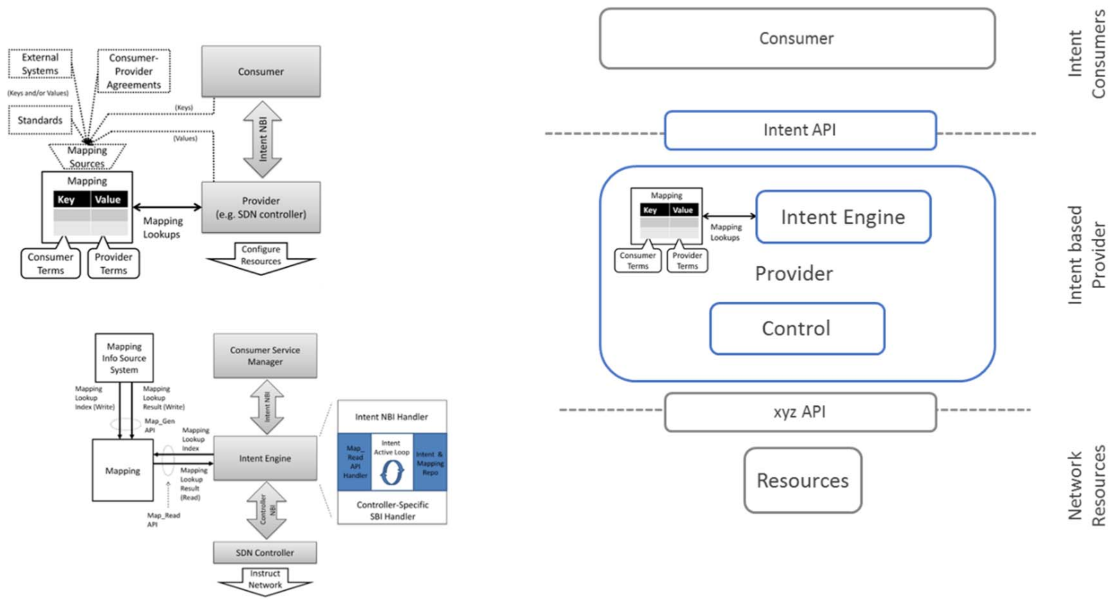

Figure 6

The Intent API is differentiated from the engine behind. The Intent API is used to call a more complex engine, as described in subsequent clauses.

4.3.4.2 Intent-based networking systems

Gartner said in 2017 ([i.15]) "*What's new is that machine learning algorithms have advanced to a point where IBNS could become a reality soon. Fundamentally, an IBNS is the idea of a network administrator defining a desired state of the network, and having automated network orchestration software implement those policies*".

In fact, today's Intent Based solutions are often domain specific, such as for SDN. So an SDN controller may be seen as a powerful autonomous expert for SDN based connectivity, offering its Intent-based API to an E2E orchestration.

4.3.4.3 Intent-based Network Service orchestration

Intent-based Network Service orchestration is agnostic of network technologies, may serve hybrid network services, for example connect NFV and PNF thru SDN etc. IBNSO applies above Intent-based principles and allows control of resources, to orchestrate and manage dynamically changing Network Services. It allows composition of network components and scaling.

More than just connectivity, IBNSO provides also the assurance which controls the quality desired with the intent. It creates a closed loop by combining the stacks, potentially based one single data lake.

Most solution on the market claiming to support IBN follow this approach, while they typically use classic hierarchical modelling approach, with different architecture components for model, workflows, policy management, and assurance capabilities.

4.3.4.4 Intent-based Service orchestration

Intent-based service orchestration is agnostic of what service it is managing. It is driven by the business needs, less by the network needs. The approach differs from all above, as its first aim is composability of digital services, for whatever need. The scope can be beyond pure Network Services, but does not exclude them. By nature an IBSO is not specialized on anything, while it allows to integrate with any specialized sub-ordinated orchestrators or controllers, management systems or resources directly.

Looking at the principles Lenrow stated, beyond invariance and portability, IBSO is based on composition of (any) services, and it uses context (expressing the why - "everything has a purpose") to identify the needs for something, what also avoids policy conflicts.

The model catalogue of an IBSO provides declarations (descriptions) of service objects and their relationship, with policies to define the behaviour with context, executed at runtime depending on the current status by an intelligent engine - truly "model-driven". Each object has sort of autonomy based on its policies declared with its descriptor, which decides about instantiation and composition at runtime. No need to define a whole process at a single place. The precondition for agile compositions. It requires an Intent-based modelling as described above.

IBSO also allows a simplified architecture, which uses one service model to define the intent graph (aka service model) including policies and closed loop, all executable by one generic engine, in a choreography with all architecture components specific for the service (e.g. controllers, management systems, resources, etc.).

Service agnostics means also industry agnostic at the same time. A broadcast company, an automotive manufacturer, or a hospital, they all benefit from IBSO, as they can order their applications, with specific configurations, directly connected from a private cloud running on an edge, assembled and orchestrated as services.

4.3.5 Proof of concept

4.3.5.0 "Intent-Based" is proven

Above concepts have been developed and evaluated by different organizations, operators and vendors. A few are named here, to show evidence the concepts work and have been proven.

4.3.5.1 Intent API

As mentioned, ONF has defined the basic concept for Intent API [i.17]. In fact, many SDN controllers provide an Intent API and allow an automated SDN routing based on declarative policies. Within an architecture it may be shown on resource management level.

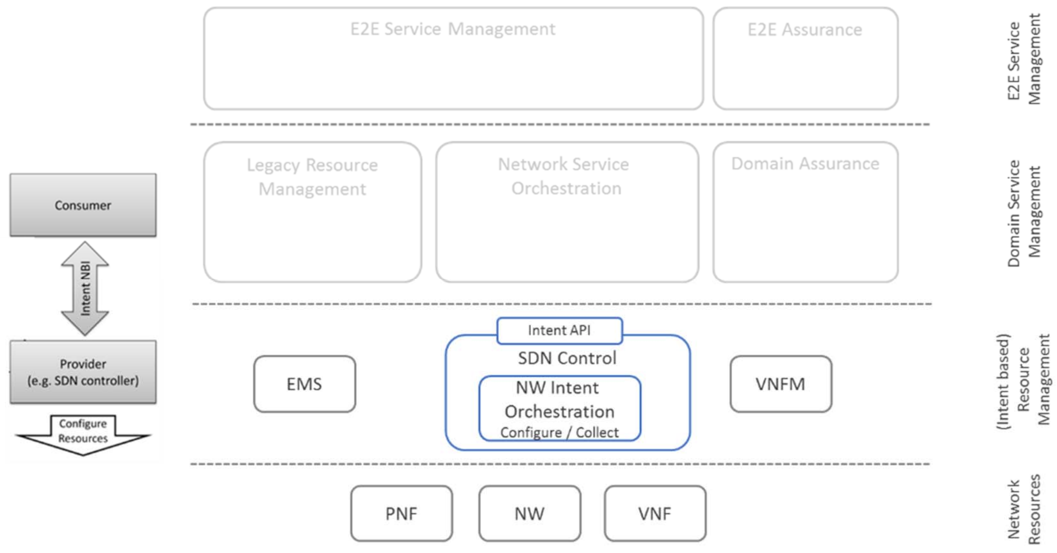

Figure 7

4.3.5.2 Intent based networking system

The NeMo project [i.18] can be seen as a concrete implementation of an IBNS, with an Intent API and an Intent engine. It is using Intent-based policies to create virtual networks comprised of nodes with policy-controlled flows. It addresses a domain specific approach for Intent-based SDN and NFV based networks.

Its scope is on Network connectivity, and it provides an abstraction model defining specific keys, such as Link, Node, Data, Connect, Disconnect, Notification or Policy. Its southbound focus is on the SDN controller.

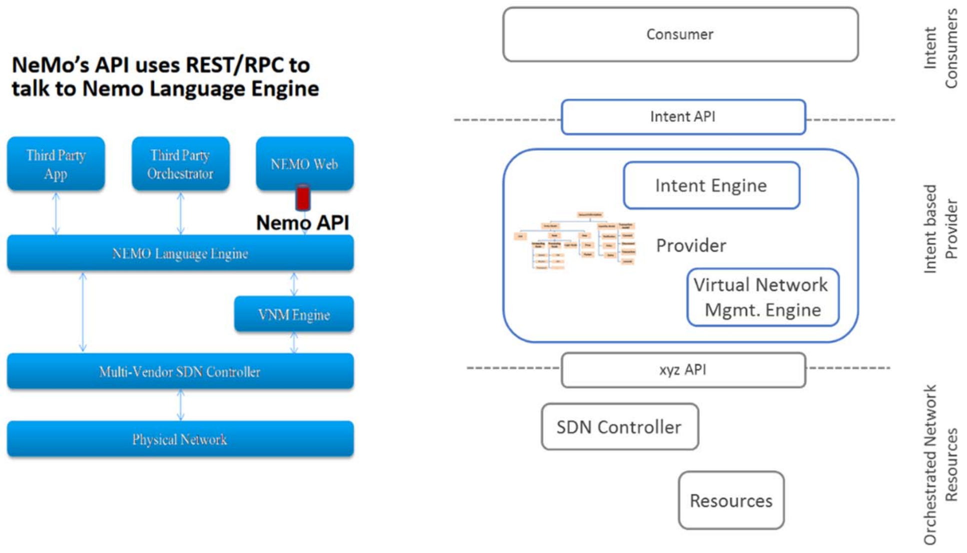

Figure 8

4.3.5.3 Intent based Network Service orchestration

A wider scope of intent based orchestration is reached by many vendors in the meantime. It is not specific for any network technology, but primarily addressing purpose of connectivity in virtual networks with virtual network functions. Figure 9 indicates on the left side a high-level approach suggested by [i.19]. The right side puts it into context with other Architecture components, such as an intent based SDN controller.

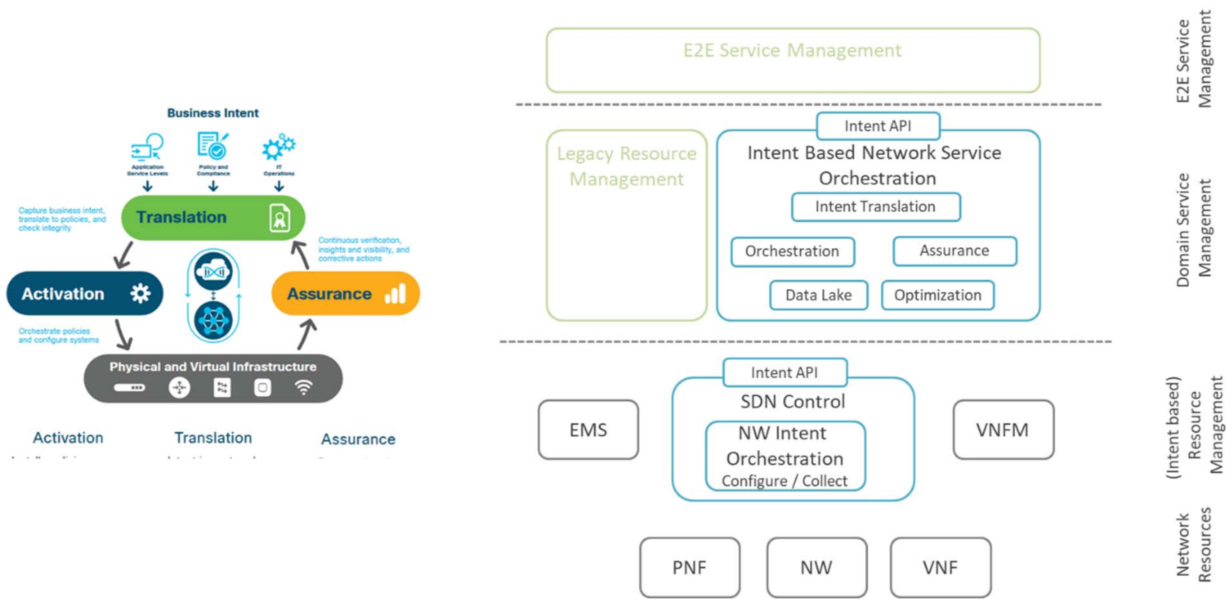
Business Intent
Application Service Levels
Policy and Compliance
IT Operations
Capture business intent, translate to policies, and check integrity
Translation
Continuous verification, insights and visibility, and corrective actions
Activation
Assurance
Orchestrate policies and configure systems
Physical and Virtual Infrastructure
Activation
Translation
Assurance
E2E Service Management
Legacy Resource Management
Intent API
Intent Based Network Service Orchestration
Intent Translation
Orchestration
Assurance
Data Lake
Optimization
Intent API
SDN Control
NW Intent Orchestration
Configure / Collect
EMS
VNFM
PNF
NW
VNF
E2E Service Management
Domain Service Management
(Intent based) Resource Management
Network Resources

Figure 9

4.3.5.4 Intent-based Service orchestration

The holistic approach of IBSO is represented with [i.20] and was published in [i.21]. The service model works as a solar system, driving provision, activation, assurance and closed loop. It allows to define business services (CFS) defined by product management, and allows a composition based on underpinning technology domains or resource management and applications, using legacy or NFV/SDN based infrastructures.

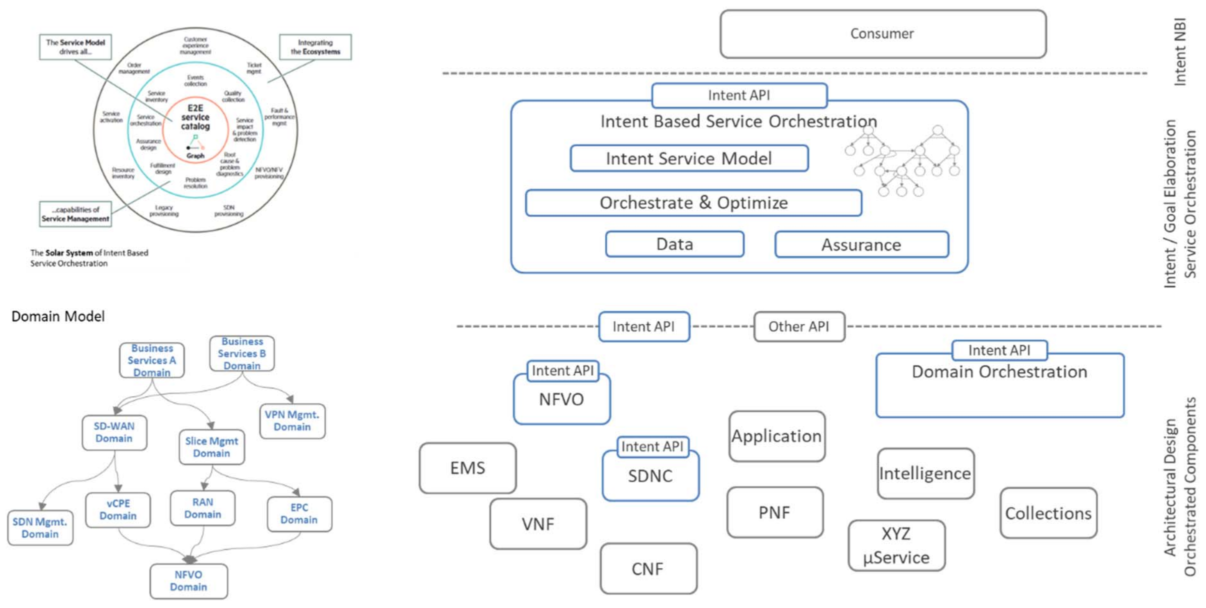

Figure 10: Intent-based Service Orchestration across Domains, driven by Intent-based Service Models

Clause 4.4 describes the mechanisms, features and implications of a consequent Intent based Service Orchestration.

4.3.6 Relevance for ZSM

All four above mentioned implications are applicable for a ZSM architecture:

- Intent API (IBA).
- Intent-Based Networking Systems (IBNS).
- Intent-Based Network Service Orchestration (IBNSO).
- Intent-Based Service Orchestration (IBSO).

The Intent API should be provided on different levels: Network Controller, Domain Management, E2E Service management. Or even across Operators (see MEF LSO interfaces).

Within the managed network Intent APIs are desirable but not mandatory, e.g. as Controller APIs or Resource APIs (network controllers, firewall, router, etc.).

Intent-based networking systems can be applied as specific controller functions, such as for SDNs. They serve ideally to an Intent-based network service orchestration.

Intent-Based Network Service Orchestration (IBNSO) may serve on E2E orchestration level, as well as within a domain, such as for an EPC or IMS. Depends on the organizational scope and the spawn of the managed network services. It may go across different network domains. IBNSO may combine fulfilment and assurance in a combined solution, the closed loop will be realized using declarative policies combining dynamic fulfilment with cognitive assurance.

Intent-based service orchestration may start on the CFS level, or even on a product catalog and order management level. It allows to design services from product management point of view, refining services using goal based declarative policies, reaching down to Intent-based APIs of domain orchestration. Domain orchestration may also use an IBSO approach, and control resources at the lowest level. Like IBNSO it allows to combine fulfilment and assurance in one orchestration engine, supporting service problem detection and resolution, all defined in one Intent-based service model.

4.4 Means of Automation: Intent Based Service Orchestration

4.4.0 Introduction

The use of Intent-based technologies is identified as a means for automation. However, the notion Intent addresses several aspects, in various ways, to address various problems.

Intent based Service Orchestration make use of all aspects of Intent based in its full consequence:

- Intent API.
- Intent-based management.
- Intent-based modelling.
- Intent-based model-driven generic orchestration.

See also clause 4.3 to differentiate the different aspects of Intent Based.

Intent-based modelling allows an "Intelligent software to understand the user goals" (as Dave Lenrow says [i.16]). The herein described means for automation explains very specific, how intent-based modelling enables business orientated and agile service offerings, with a zero-touch management, agnostic for managed technologies.

Other means of automation, presuming intent based service orchestration can be combined:

- Closed loop modelling (with intent means).
- Combine fulfillment and assurance in one orchestration engine.
- Automate integration of Intent based API.

All of this is driven by the Intent-based modelling, as described herein.

4.4.1 Motivation

4.4.1.1 Business Goals

The high level business goals of operator's looking at future operations can be summarized as follows:

- Increase service agility: accelerate time to market for new or adapted services.
- Increase operational flexibility: support organizations as needed.
- Reduce cost of infrastructure and operations: focus on automation ("zero-touch") and simplified, agile service development.

These goals can be measured as:

- reduction of time to market: from months to days;
- agile development with a fail-save approach: e.g. 4 releases instead of 1;
- zero touch provisioning: from days to minutes;
- zero touch assurance: "closed loop";
- focus DevOps on service design, not on integration developments.

The rapid increase of new service types and infrastructure technologies is driving complexity and blocking these high level goals. Software-defined automation and zero-touch operations is the right answer to this challenge. However, standardization for service and network management evolves too slowly to cope with market needs. An agile approach is required.

4.4.1.2 Inhibitors of agility

Service management architecture is often implemented as monolithic siloes, fixed component stacks. Typical example is the split of assurance from fulfilment, which have completely independent lifecycles. Another example is the separation of NFV-based from classic network services, which makes hybrid service offerings very difficult and static. In addition organizational dependencies, existing ownership and scope of domains and management systems make operational flexibility very difficult.

Defining integrations and data models in standards need to accelerate integrations. In fact, each integration still needs a lot of adaptions, for various reasons. The introduction of an additional software, for example to automate a traffic path optimization as a small software module (like a micro-service), is difficult, as the standards consider a fixed application map, rather than a dynamic management software landscape.

In the meantime more open and innovative OSS initiatives started more agile approaches, primarily using cloud and micro-service technologies. However, service modelling and orchestrations still has a rather classic structure of hierarchical telco services, not making use of more dynamic software elements.

4.4.1.3 Cope with complexity

One goal of standards is avoiding complexity. It can be differentiated:

a) Complexity of management systems architectures versus.

b) Complexity of services to be managed.

Telco standards (and new open source approaches) around operations (TMF, ETSI NFV, MEF LSO, ONAP) address the first. Most of standards assume, that future services and infrastructures can be mapped to a once agreed landscape of functional component stack. This may work for big and generic software functions.

Actually services will become even more complex and may require additional software functions. Just NFV brought a new layer in the layered stack and resulted in a new platform called MANO. MEC allows more dynamic infrastructures and platform managers. 5G slicing is an orthogonal dimension on top of operator underlying core network services. B2B and IoT services will explode, while sitting on very dynamic infrastructures.

Looking at this, any well-defined architecture today, may not work tomorrow.

A way out of this dilemma is to allow more autonomy of smaller modules, i.e. higher and more consequent modularization: a single module may know nothing about the customer service it is finally supporting, nor does it know about the infrastructure it is running on. Orchestration becomes choreography. But the service modelling need to allow for this.

Intent based modelling and orchestration is a key approach, to cope with complexity. Future management systems need to support service evolution and incremental growth, without breaking existing systems. This can be achieved by composition of small autonomous service components.

4.4.1.4 Simplify automation

A technical inhibitor to achieve the necessary agility in time to market today is simply the number of systems to be touched. Is it possible to simplify the design and implementation of a service using one common means, such as One-Model-Driven?

Furthermore is it possible, based on today's technical software and API technologies, to automate the integrations independent from a given data model? Integrate new software like a part of a service?

Last but not least: is it possible to avoid coding workflows? Can a generic engine evaluate a new *consistent* state of a complex service instantiation, just based on given intent (or goal), and generate the runbook to adapt at runtime?

4.4.2 Problems to be solved

4.4.2.1 Aspects to consider per service

The support for new services has to be spread across a set of OSS components:

- Service models (in catalogues and inventories).
- Workflows (in orchestration engine).
- Policies for fulfillment (feeding orchestration engine, e.g. what configuration, vendor, version etc. in what situation).
- Policies for assurance (in fault, performance, quality and problem management, e.g. when is a service good enough, when does it have a problem).
- Policies for closed loop (part of assurance, combined with fulfillment, e.g. how to solve what problem).
- Vendor specific configurations may interfere with operator specific configuration.
- Perhaps other places, etc.

The notion "Policies" is often a bunch of very different means, to control behaviour at runtime. In classic OSS systems they vary from application to application.

Workflows use policies for decisions at runtime. They may be specific per service model and associated change scenarios. The number of possible workflow permutations, looking at a chain of n network functions can be expressed like:

$$n \text{ Network Functions} \Rightarrow (n^2)^{m} \text{ Workflow Permutations}$$

where m is a product of factors like NF SW Versions, Mix of PNF and VNF, product bundles and releases, Different operational requests (e.g. mass upgrade), separate fulfilment and assurance needs, etc.

Each service design results in an exponential number of possible workflow variants.

Furthermore, the design complexity of a single workflow variant also increases exponentially with the number of service components. The designer has to specify conditional statements for each component separately but also for the combination of those. The testing phase also requires dedicated workflows to thoroughly test all service conditional statements possibilities. If the service fails in production, the faulty logic has to be identified within the workflow, fixed and tested again.

4.4.2.2 Behaviour is independent from service model

Policies and workflows to control the provisioning, assurance and resolution are today defined at other place and using other means, than the service model. At development time, model, workflows and policies need to be kept consistent: any change in the model may follow many changes in policies and workflows, at various places. Still the system can cope only with expected scenarios. This is the biggest inhibitor to achieve agility, flexibility and reduce costs.

4.4.2.3 Models are independent from software

While service models today are agnostic of the software how they are implemented (top down approach), an existing software may offer APIs to specific services, which may enrich the abstract service model (bottom up approach). If APIs could be translated into service descriptors, new service compositions would become a matter of service design, relationships and model interaction, instead of integration developments.

The classic concept of CFS/RFS/Resource becomes blurry: the resource of a service provider, becomes a "CFS" for the infrastructure provider. Hence what a CFS/RFS or Resource is, become just a matter of scope and perspective. The difference should not be relevant for an orchestration.

4.4.2.4 Organization and vendor independence

An essential problem are dependencies of organizational structures, e.g. network or service domains. Domains have their own scope and means of management software. That limits the flexibility to restructure services as well as organizations, or replacements by external suppliers. If the organization's deliverable can be integrated like XaaS, composed into the higher service, it is easy to adapt and replaced, when needed.

Vendors often require vendor specific configurations with EMS, e.g. to provide additional automation. Also Operators desire specific configuration of services. In existing approaches this is often difficult to split. In fact both can provide a huge value and should be supported, e.g. with a more granular service modelling.

4.4.2.5 New modelling approach

A less hierarchical and more granular modelling approach can support higher dynamics of service composition and software modules:

- Split service component composition for various purpose (e.g. VNF deployment, VNF vendor specific configuration, VNF operator specific configuration, Service configurations on VNF).
- Easy shaping of organizational domains, e.g. order management, customer service management, network domain, XaaS suppliers, use of "platforms", etc.

Given the above mentioned OSS capabilities can be combined in one way of modelling, a huge step towards simplification is achieved:

- One model, to define:
    - Service composition.
    - Provisioning behavior.
    - Quality detection.
    - Closed loop behavior.

- Able to scope on:
    - Organization or network domains.
    - Model XaaS.
    - Infrastructure specific descriptions (NFV or classic).
    - Vendor specific configuration descriptions.
    - Operator specific configuration descriptions.
- Able to drive a generic engine:
    - To take decisions at runtime.
    - To generate a runbook (workflow of actions) at runtime.
    - Based on as-is-status, to achieve a to-be-status.
    - Even for unplanned or unexpected change scenarios.

Today's available languages do easily or most optimal allow to combine service structures with behavior. Service models are documenting a desired service structure. Semantics and reasoning to drive an "intelligent" orchestration engine is missing. The engine need to know an algorithm (workflow), which needs to match to the specific model, and combined with various policies. The consistency and number of workflows becomes an exponential problem over time.

4.4.3 Principles and Concepts

4.4.3.1 Intent based modelling

An intent based modelling allows the replacement of workflow engine, by a generic "intelligent" engine. An "Intent" is the purpose of something. No service (or resource) exists, without another service requesting it (the "intent"). Vice versa, a service (or resource) is deleted, as soon as its "intent" service is deleted. The intent declares the WHAT and the WHY. The "intelligent" engine detects the necessity of actions.

The model description needs to capture the entire semantics of the model, not just the data structure. It can be called the reason behind the model. If the reason is not expressed, the "intelligent" engine would not be able to build the **prescription** and hence unable to adapt the system to the new situation. To gain flexibility with changing situations (e.g. service variants or component status'), it is important to express the **policies** taking a current context or status into account, within the service model description. This way, the orchestration engine can use these policies to figure out the appropriate deployment of some infrastructure or network combined with NFV combined with "what-ever" under given circumstances.

Classic workflow orchestration can only prescribe, what is coded. Intelligent orchestration using an intent based model is able to generate the prescription - the HOW - at runtime.

4.4.3.2 Modelling language: Dynamic Service Descriptors (DSD)

Service model and behaviour is described as a set of service descriptors, which declare the service objects, their relationship and its behaviour policies. They can be called Dynamic Service Descriptors (DSD) ([i.20], [i.21], [i.23]).

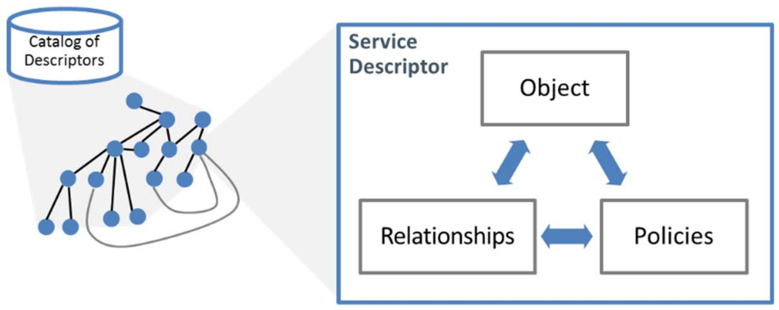

Figure 11: Dynamic service descriptor concept

Relationships go beyond a classic definition of service hierarchies, and allow a (directed) graph: each service object is requested by other service objects (the WHY) and may request to other service objects (the WHAT).

At runtime, each service instance has a well-defined status (MTOSI TMF518_SA_2). Transition of the status can be associated to a concrete action request of another management or a managed system.

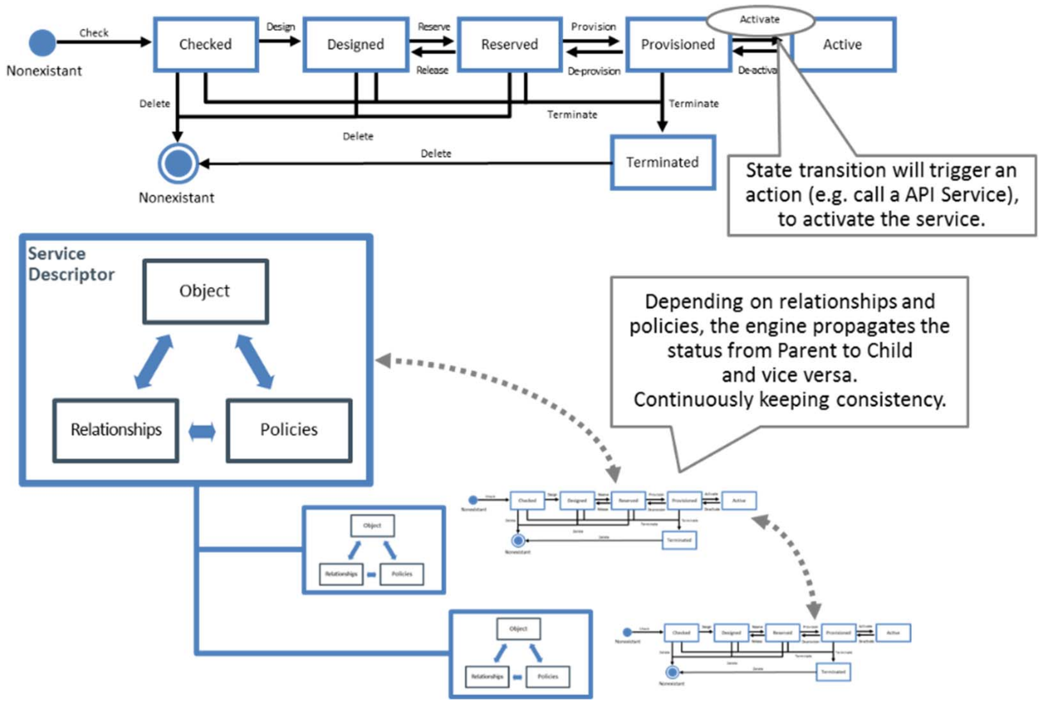
Check
Nonexistant
Checked
Design
Designed
Reserve
Release
Reserved
Provision
De-provision
Provisioned
Activate
De-activa
Active
Delete
Terminate
Delete
Delete
Terminated
Nonexistant
State transition will trigger an action (e.g. call a API Service), to activate the service.
Service Descriptor
Object
Relationships
Policies
Depending on relationships and policies, the engine propagates the status from Parent to Child and vice versa. Continuously keeping consistency.

Figure 12: Dynamic state transition of service descriptor instances

DSD allow Policies of various types, which can be differentiated:

- By relationship, e.g.:
    - An underpinning "resource"(-service) needs to exist, before its parent(-service) can be instantiated.
    - A contained "component"(-service) can only be created, after its parent(-service) is instantiated.
- By functions, e.g.:
    - Set parameters using specific functions and expressions, e.g. using parameters of neighbor services, requesting external input, calculating state propagation, etc.
    - Identify related nodes at runtime, such as children, parent, siblings, ancestor, etc.
    - Graph-instantiation may use functions such as if, then, and may decide e.g. about the version to instantiate.

- By explicit declaration, e.g.:
    - Define a policy as a service component, with an own service descriptor, e.g. "best price", which evaluates a current status from an explicit algorithm or software.
    - Define a policy within the service, e.g. a VPN service asks for a minimum bandwidth.

In addition DSD have policies controlling the engine's behaviour, such as for rollback or error handling.

4.4.3.3 Modularization and composition of services

As learned monolithic systems of service hierarchies are not helpfull, rather more individual service objects with relations as a bigger information model need to be built. Semantics of the individual service objects need to be well defined, making sure they are rather independent of the context they appear in. Thereby what goes on each node of the information model, is independent on where that node will get deployed. For example an MPLS VPC service descriptor should be agnostic, if it is part of a customer VPN or an EPC backbone. Hence the approach allows incremental models which can grow with the evolution of the models. It allows to add things in without breaking what is already there.

A classic information model has a "spaghetti-structure", you need to always re-evaluate everything as something could have changed in a completely different corner of the world due to the change. The intent based modelling allows an "intelligent" software to find out itself, what is needed to change.

4.4.3.4 The generic orchestration engine

With intent based modelling and declarative service descriptors (such as DSD), it is possible to build a generic engine, which generates and orchestrates actions to change the service's state. The engine decides the necessary state transitions and finds the necessary actions.

Within the descriptor, a state transition is associated with an action request, than can be a call to a subordinated management systems (NFVO, SDNC, EMS, etc.) or to the managed systems directly, e.g. to deploy or to activate a service. The sequence of actions is generated at runtime and hence also able to cope with unforeseen situations, as long as a consistency can be found in the instantiated graph. The problem of exponential growth of workflows becomes linear: a single more descriptor just needs a single set of state transition actions.

The orchestration engine executes the following steps to build the final runbook.

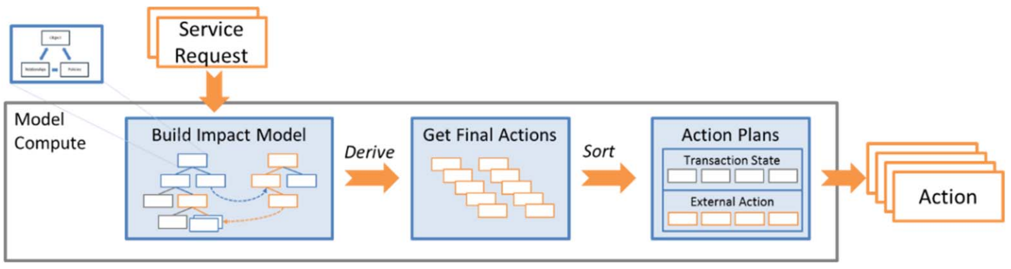

Figure 13: Engine computing the runbook

Thanks to its capability to build an impact model triggered by any inconsistency, the engine is able to cope with various situations and change scenarios, even unforeseen changes.

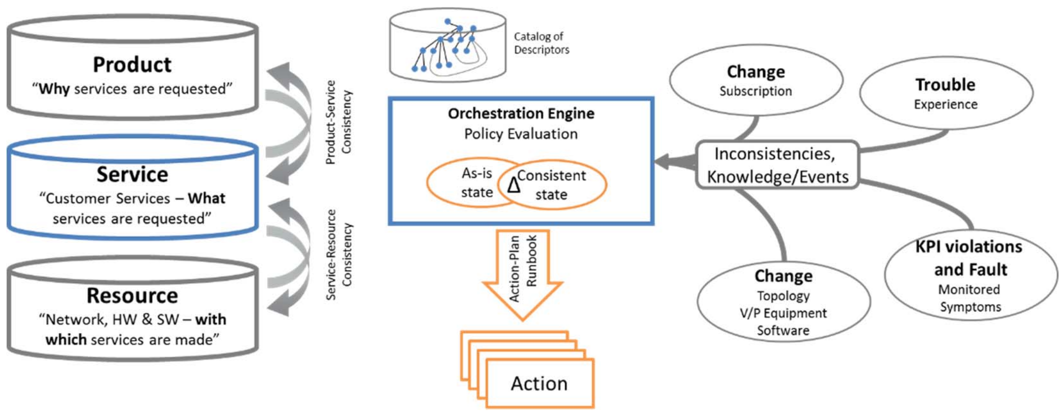
Product
"Why services are requested"
Service
"Customer Services – What services are requested"
Resource
"Network, HW & SW – with which services are made"
Product-Service Consistency
Service-Resource Consistency
Catalog of Descriptors
Orchestration Engine
Policy Evaluation
As-is state
Δ
Consistent state
Action-Plan Runbook
Action
Change Subscription
Trouble Experience
Inconsistencies, Knowledge/Events
Change Topology V/P Equipment Software
KPI violations and Fault Monitored Symptoms

Figure 14: Policy evaluation defines service consistency and generates necessary runbooks at runtime

The engine computes itself the desired consistent state, by building the impact model. This is different from typical desired state engines. Then it finds the necessary actions (per descriptor) to turn the inconsistent as-is state into a consistent state again.

4.4.3.5 Automate integrations and Service marketplace

Finally service descriptors and associated actions can be generated as code, from to machine readable APIs, such as Swagger with REST for OpenAPIs^{TM} (see https://swagger.io/docs/specification/about/). That allows a fastest integration e.g. of new network functions or management software into a service model and its orchestration. Making such packages available thru a marketplace, allows new models of supply chain, for most cost effective integrations of new services.

NOTE: OpenAPI Specification and OpenAPI Initiative and their respective logos, are trademarks of the Linux Foundation.

4.4.4 Implications

4.4.4.1 Applicable languages and necessary extensions

Modelling languages investigated are TOSCA, YANG and HOT. All of them have lacks, when it comes to automation needs. The DSD language concept requires extensions features in two ways:

- For Service *Model* Definition (some language support them partially):
    - Incremental multi-level graph modeling - the most important feature (others allow e.g. monolithic trees, graph templates, or limited layers).
    - Full multiple inheritance - enables e.g. to compose new service from different existing in one descriptor (others may not support inheritance at all, or only simple inheritance).
    - Support for loose coupling between compound services and the service components - the difference between "kind" (comparable with Java interface) and "service" allows adding multiple implementations of the same kind of service over time.
    - Packages/name-spaces - allows structuring beyond just naming.
    - Catalog versioning - allows automation of upgrades at runtime.
- For Service *Automation* Definition, that means the declaration of business logic using policy expressions or other features (not supported any language):
    - Semantics definable within language (Parameter policies, decomposition, auto-creation) - most important feature to control the engine (no other language allows this).

- Telco Grade Deployment State Model - the state model of instances uses MTOSI TMF518_SA_2 as basis. State transitions are defined as operations in the descriptor (e.g. change from provisioned to active calls a micro service).
- Automatic dependency calculation - dependencies may be relevant or not at runtime, the engine calculates the relevant dependencies based on the given policies.
- Dynamic service graph creation - based on dependency calculation, the instance graph is dynamically evolved as needed (other languages support in best case static lists of relationships).
- Automated Modification of existing services - depending objects need to change to the same status as the intent object (controlled by policies). If an object exists already, but has a wrong status, it is automatically modified.
- Automated upgrade to new service model versions - each descriptor has a version; multiple versions of the same descriptor may exist. The engine performs automatic upgrade of am instance if a later version exists (depends on the level of version difference).
- Access to Resource Inventory - related with state transition operations, such as "designed" to "reserved", which causes an inventory update in case the resource is managed there.

With these features listed above, it is possible to specify the structure and behavior of a service as well as its relationship to its environment in a code-less declarative way.

4.4.4.2 Architecture simplification

Simplification of architecture does not necessarily mean less components. But the composition and use of whatever needed components need to be simplified.

The generic orchestration engine described above, indeed simplifies the orchestration platform as it combines service catalog, service inventory and orchestration engine, and activation actions, in one engine.

Such an orchestration platform may be instantiated more than once, any may serve different scope of the model, e.g. service orders, customer facing services, specific resource facing services per domain (e.g. SDWAN or MPLS), with different vendor technologies underneath, assurance services. The designer may decide about the coverage of the platforms and the scope of models.

With that approach, advanced architecture and deployment blueprints like TM Forum HIP concept (see [i.24] and [i.25]), as well as a MEF LSO concept (interacting LSO instances [i.26]) is easy to implement.

The orchestration platform does not fix requirements to an underlying component stack. Moreover, it allows orchestration of any additional automation software. Any desired architecture of platforms, management applications or automation software is possible.

Figure 15 is motivated by TMF HIP deployment blueprints ([i.25]) and illustrates, how a service model can be composed, and how it can be mapped to individual service platforms.

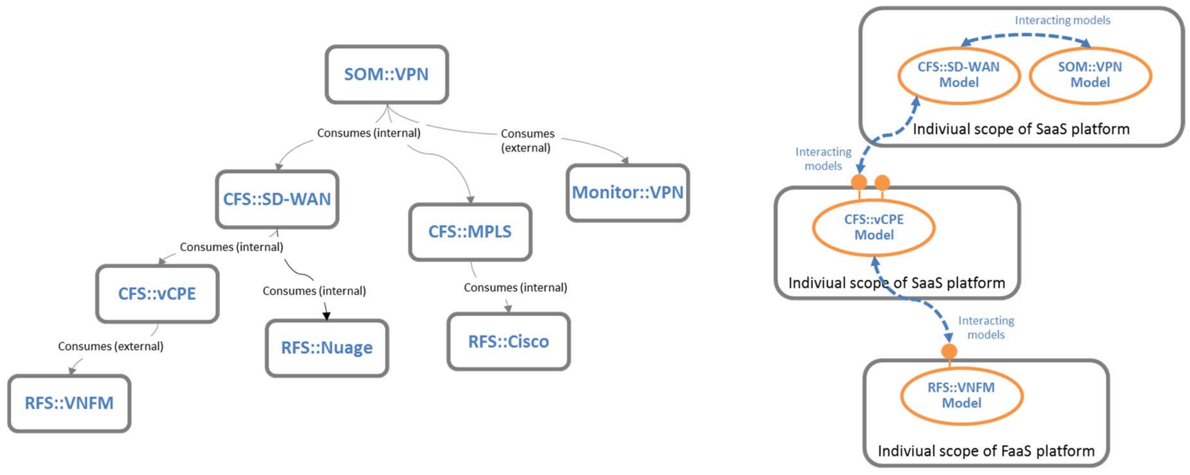

Figure 15: Service Model across individual HIP platforms

However, in existing operator's solutions it is often a very typical stack of subordinated resource management systems (controllers, VNF manager, NFV resource orchestration, element and network managers, direct interaction with managed resources, assurance services, probes, etc.).

4.4.5 Proof of concept

The orchestration concept as described has been used to address various use cases:

vCPE + SD-WAN:	L2/L3 VPN across SD-WAN with Value Added Services provided over VNFs on NFV, connected thru SDN across Datacenter/Cloud and customer premises.
Hybrid SFC:	provisioning of service function chains across PNF/VNF, e.g. coupled with cloud based business applications, networking thru SDN, VPNs, e.g. for VPLS, L2/L3 VPN, Sonet or MPLS replacement
Mobile Core:	provisioning (thru NFVO) and chaining (thru SDN controller) and configuration of mobile core network functions, e.g. PCRF configuration for serving residential customer services such as Triple-Play or TV.
Application Orchestration:	Structured in two parts: - Application Lifecycle Management using NFVO. - Application Configuration Management (e.g. using Ansible) based on Service needs.
Service Order Management:	Modularization of Service Order Management from Service Provisioning, by interacting models. Allowing combination of classic (existing) processes together with automated services.

Further use cases are considered in current PoCs:

Network Slicing:	provisioning of network slices, as a specific service function chain per customer or application, across multiple network services of the operator.
Mobile Edge Computing:	Integration of Mobile Edge Application Orchestrator (MEAO) with (intent based) Service Orchestration thru model interaction, hence allowing modelling and automated orchestration for MEC CFS with MEC user applications at ME systems.

The concept is proven in operation, many operators are using the approach today or started introducing it. Table 1 shows only concrete implementations; more exist in PoC trials.

Table 1: List of operators using intent based orchestration with DSD

# Cases existing or ongoing solutions	Services
1	vCPE enterprise services, with self service portal, 100% automated provisioning and operation, some extended for WiFi and LAN-Management Services
1	Fully Automated migration of classic PNF based CPE to virtual VNF based vCPE services.
6	SDWAN with vCPE enterprise services. Combining standard NFV MANO at telco cloud, with SDN across VNF, PNF and customer premises.
1	MPLS based VPLS using classic PNF routers (e.g. Cisco)
1	IP-VPN and FTTP based provisioning
1	Residential Services (Voice, Internet, TV) over PNF
1	Multiple B2B type services (VPLS, MBB, Unified Communications, Security, ...)
1	Mobile Core Network Services on top of NFV MANO
1	Full LTE process automation
1	End-to-End orchestration of multiple domains, control of connectivity- and cloud-services, acting as a service broker.
2	Provide Order and Service Orchestration with interacting models, fixed line and mobile

Operators using the approach name the results:

- Time to Market for new services: from 9 months to 10 weeks.
- Service Instantiation: 30 days to 5 minutes.
- Significant cost savings.
- Fully integrated end-to-end Orchestration.
- Deep degree of automation on the provisioning of MPLS nodes.
- Foundation of the OSS Transformation that will change the way fulfillment and orchestration.
- Flexibility of the technology (including UI) to allow agile delivery and short reaction times.
- Very fast upgrades of big network releases becomes a matter of weeks instead of months.
- Integration with NEs and other systems using different protocols within days.
- Reduce the number of incoming calls (Call Center) and reduce MTTR.
- The orchestration blueprint is very successful and ideal for global deployment.

The solution was also shown also in a catalyst 2016 (see [i.22]) as a new OSS concept for a policy driven "operations-defined automation".

4.4.6 Relevance for ZSM

4.4.6.0 Several implications become relevant

The described concept of an intent based orchestration allows a number of implications to the targeted ZSM framework.

4.4.6.1 Define Everything as service

Enable a consequent paradigm change, that everything is a service: instead of a fixed definition of what a CFS, RFS, Resource can be, allow to model everything what a sub-systems may provide as services, i.e. network functions, network management systems, controllers, platform managers, element managers, applications, monitors, fault management, performance management, etc. in consequence, make their services a part of the service model.

4.4.6.2 Operators landscape is a service

Previous principle includes also the operator's landscape of management systems (also known as OSS). The intent based information model includes modeling and management of management systems themselves like any other service. This allows not only orchestration of OSS application (scaling, etc.), more important it allows to relate customer services for example with monitoring system setup.

4.4.6.3 Closed Loop is a service

The previous principles encourage modelling of closed loop as native part of the service model. Service models can be defined to manage the setup of probes and monitoring systems, configuring the collection and propagation into quality indicators associated with the CFS/RFS, and hence manage service level policies and problem resolution actions. The described concept has been already used to align the complete assurance setup and its lifecycle, for problem detection and resolution, with in one model together with service provisioning.

4.4.6.4 Support brownfield and classic services

NFV and SDN infrastructures will be dominating in future, however, virtual networks such as MPLS, VPN etc. will be used in combination or as underlay. Any future ZSM framework needs to consider other technologies than only NFV and SDN. Another huge advantage is the ability to automate migrations from classic to NFV/SDN.

4.4.6.5 Support unexpected future services

While future service cannot be known, there will always be a structure of dependencies, how to build them, and new software to configure them. 5G Slicing and MEC is just the next step. Intent based modelling and dynamic service descriptors allow to model any service of today and tomorrow.

4.4.6.6 Revisit modelling languages

As described, existing modelling languages have very limited features for a fully intent based model driven orchestration, without additional coding, policy management or other implementation needs. Intent based modelling with the DSD concept as described here, allows the combination of intent, service structure, and behaviour, combined with use of existing software API in a one-model-driven approach. But it requires extensions of today's languages.

4.4.6.7 Define an architecture framework

The ZSM framework will enable any service management, using any sort of resources, allowing various management software components. Therefore the ZSM framework enables composition of services, using applications and data models as needed for desired services. It will provide guidance to build and integrate new components based on principles of cloud-native apps, machine-readable Open APIs and micro-service technologies.

First priority is agility providing new customer services, as this makes the business case. Not to confuse with building new software. Only a fraction of new services, require new software. The ZSM framework can provide guidance for a more agile CI/CD process for new services, using intent based service design and orchestration engine.

Second it will guide best practices for efficient use of software technology (cloud-native, micro-services etc.), which allows to build and integrate necessary micro-services, less fixing what micro-services are needed.

More generally ZSM is not about fixing "yet another architecture", rather it is desired to provide guidance on how to build service-oriented and modular architectures to accommodate composable services. See also TM forum ZOOM projects [i.24], such as ODA, HIP and Future Mode of OSS.

4.5 Means of Automation for Network Governance

4.5.1 Motivation

Operators are changing their vision on current management approaches motivated by the increased operational costs implied by the complexity of large infrastructure management [i.27].

Future network infrastructures should be highly adaptive and autonomous, stressing further the dynamic relationships and operations of the involved resources. Some functions that were traditionally performed by management systems will be autonomously carried out by the network itself.

Operators will be mostly settled towards decision-oriented tasks. What these decision-oriented tasks are and how they impact the decision elements are the main issues at hand. The introduction of closed control loop and autonomic capabilities generate a re-assignment of tasks previously carried out by humans, which will now rather focus on future network operation and planning, rather than continuously monitoring the behaviour of individual components.

Several initiatives and projects have investigated or are currently investigating the aspects and evolution of network management and so-called network governance: the Generic Autonomic Network Architecture (GANA) defined by the EU FP7 EFIPSANS project [i.28] and ETSI GS AFI 002 [i.29], the FOCALE architecture defined by Strassner et al. [i.30], the *Unified Management Framework* (UMF) defined by the EU FP7 UniverSelf project [i.31], the *Autonomic Network Infrastructure* (ANI) defined by the IETF ANIMA working group [i.33], and the Linux Foundation open source project *Open Network Automation Platform* (ONAP [i.34]) to name a few. The various projects base their principles or are using a set of concepts such as Policy Based Network Management (PBNM [i.35]), or *Intent Based Network Management* (see also clause 4.3).

Similarly, the present Network Governance approach aims at describing the concepts, functionality and mechanisms (the means of automation) which can be used to govern and manage network infrastructures, how they relate to each other and how they work as a whole.

4.5.2 Problems to be solved

Building a network governance framework also faces technical challenges in five main functional areas: business language, translation, reasoning, policies, and configuration enforcement.

Network governance is meant to provide a mechanism for the operator to adjust the features of the required service/infrastructure using a high-level language. To achieve this objective a business language may be required that will help the operator to express what is needed from the network i.e. the so-called "intents" ([i.17]). Such a business language may be modelled, for example, using ontology to add semantics and enable machine reasoning on the goals.

These high-level directives need to be translated into low-level policy rules that can be enforceable to control behaviour of self-managed resources whatever their type, either a single device or a set of devices that can group their self-* features (set of devices managed by one single Autonomic Function (AF) - see also [i.5] for definition of autonomic systems).

Reasoning is also an important challenge in the scope of network governance, as it can be exploited for the mediation and negotiation between separate (federated) domains. It is thus an important capability of the governance function to support multiple, domain-specific information models to allow interoperability between semantically equivalent, but differently instantiated models. This leads to the use of ontologies for semantic fusion and reasoning, with knowledge extracted from the various sources of data and information.

Network governance is almost always interwoven with policies lying at the highest level of the so-called policy continuum ([i.36] and [i.30]). Policies specify rules that should govern the behaviour of the managed entities. In network governance, policies are required for the selection of the optimal configuration of a service and for the translation from business level entries to low-level policies.

Furthermore, configuration enforcement mechanisms are necessary to apply the decisions. First, it is required to identify concerned devices and request each of them to perform the appropriate configuration actions. Then, each of the targeted devices will translate and apply the decision. The term "configuration" implies self-configuration and includes reconfiguration and re-optimization actions. Such actions can be triggered to adjust the configuration parameters following the evolution of the network, the service, or customer conditions.

The challenge thus consists in designing and specifying a network governance functionality ([i.31]) based on a high-level language, reasoning, policies and distribution capabilities) that can:

- Work with proper business rules and policies, while connecting high-level goals and network resources to provide the operator with an appropriate, human-friendly governance interface, simple enough to be used by non-highly specialized technicians.
- Guide infrastructure behaviors while offering a service view.
- Offer mechanisms that assist the operator to express goals, objectives, constraints, and rules to ensure the desired operation of its autonomic network.
- Help to convince the operators of the benefices of adopting autonomic approaches.

4.5.3 Solution Principles and Concepts

4.5.3.0 Functions to differentiate

Network governance responds to the need for the human network operators to supervise the functioning and controlling the behaviour not only of the underlying autonomic functions, but of the management system as well i.e. the MF core blocks: Governance, Coordination, Knowledge, see [i.31]).

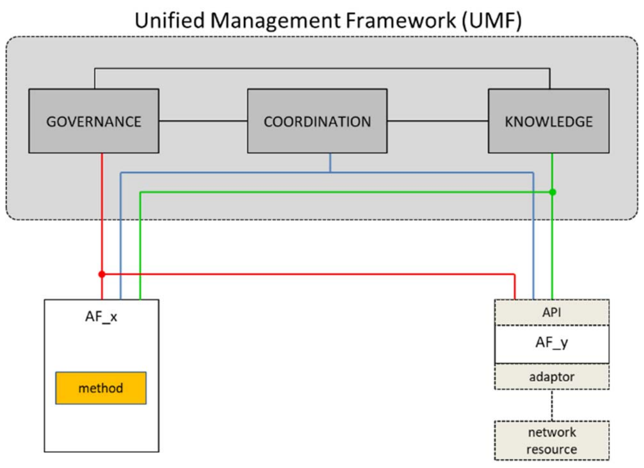

Figure 16: Unified Management Framework core blocks (Tsagkaris et al [i.31])

For a proper governance of future networks and services four main functions appear necessary (Figure 17):

- the Human-Network function;
- the Policy Derivation and Management function;
- the AF Management function; and
- the Enforcement function.

The Human-to-Network function provides methods allowing human network operators to manage the whole autonomic network, its services and the other components of the MF by editing and validating high-level objectives. A high-level objective can be defined as an overall target that the network need to achieve. These objectives are expressed in a close-to-natural language and therefore need to be translated into a language that can be understood by targeted elements as detailed in ONF Technical Recommendation (Intent-NBI, ONF TR-523 [i.17]). This translation process is implemented in the Policy Derivation and Management function, which produces low-level policies as outputs. The Enforcement function then transmits the low-level policies to the corresponding AF or MF components, while the AF Management function allows a fine-grained control of the AF lifecycle.

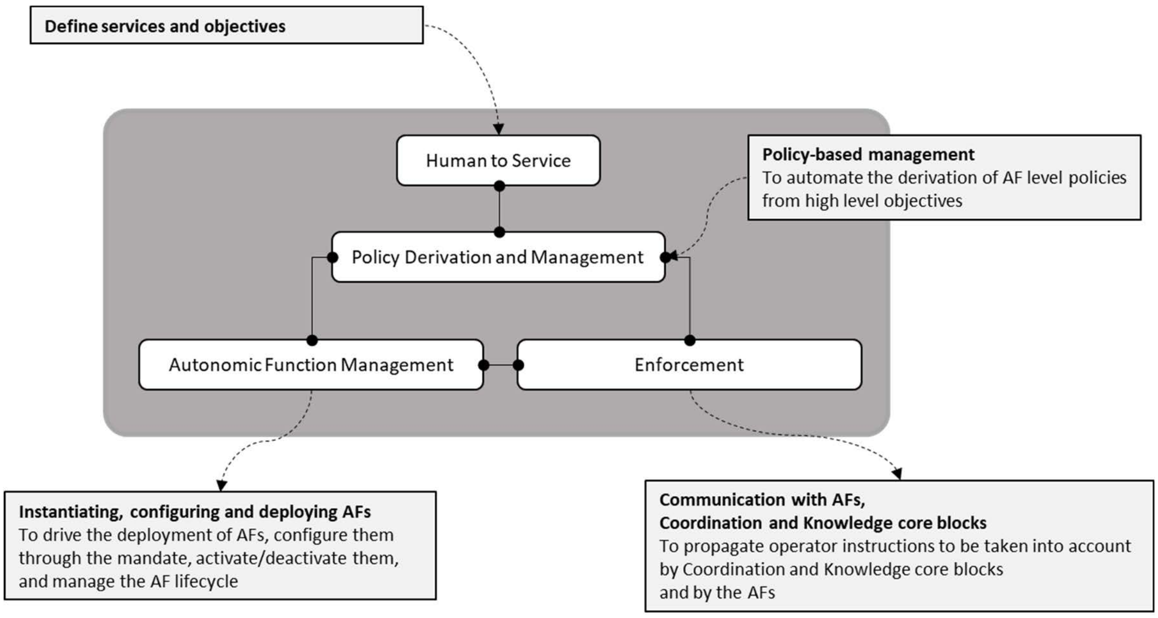
Define services and objectives
Human to Service
Policy-based management
To automate the derivation of AF level policies from high level objectives
Policy Derivation and Management
Autonomic Function Management
Enforcement
Instantiating, configuring and deploying AFs
To drive the deployment of AFs, configure them through the mandate, activate/deactivate them, and manage the AF lifecycle
Communication with AFs, Coordination and Knowledge core blocks
To propagate operator instructions to be taken into account by Coordination and Knowledge core blocks and by the AFs

Figure 17: Network Governance functional blocks

4.5.3.1 The Human-to-Network Function

The Human-to-Network function provides means for injecting input to the network through the definition of high-level objectives, and at the same time, for receiving information about the functioning of the network, services, and the MF. To reach these goals, the Human-to-Network interface can be decomposed into a set of methods: Service Definition, High Level Objectives Definition, and Supervision. The first two methods offer the programming functionality for the network, the services, and the MF, while the latter provides feedback about their status.

Service Definition

Service Definition is about the creation and representation of service descriptions. This description is later translated into terms that allow an AF to understand the requirements of individual services, such as QoS (Quality of Service), availability, or security. The definition of a service describes its invariant characteristics and their parameters.

Table 2: Service definition parameters

Field Name	Description
Name, ID and Description	Identification of the service and its Textual Description.
Resources	List of equipment types needed, with requirements for hardware, protocol and software.
AF Classes	List of AF Classes instantiated for the service deployment.
Location	Geographical area where the service will be available.
Characteristics	List of parameters describing the service, e.g. redundancy level or availability.
Characteristics Values	Matching values of the service characteristics.
Performance Parameters	KPIs used to measure the performance when the service is running (AFs will have to monitor and communicate these measurements to the Knowledge functional block).
Templates	A service template represents a type of Service Specification introduced for the purposes of fulfilment. It defines specific service values that can be dynamically referenced by multiple service instances during their lifecycle span. Several templates can be produced for the same service, corresponding to different levels (e.g. Gold, Silver and Bronze levels).

Field Name	Description
Level Objective	A set of quality goals for the service defined in terms of metrics, thresholds, and tolerances. Two types of parameters are considered here: KPIs and KQIs (Key Quality Indicators). A KPI measures a specific aspect of the performance of a resource or a homogeneous group of resources. A KQI measures a specific aspect of the service performance of a service.

High Level Objectives Definition

The Human-to-Network function provides the definition of technology-agnostic management objectives using a high-level language. These objectives are used to tune the network behaviour and the MF core blocks configuration. These high-level objectives correspond to:

- Service characteristic values: Service demands concern requests relevant to service provision, such as the delivery of a new service to a given customer with given service characteristics, the accommodation of new traffic in a specific geographical area and time period, or the modification of already active service quality conditions. Service configuration assigns values to already predefined service characteristics (e.g. redundancy level or availability).
- Configuration of MF core blocks tuning behavior of the Coordination and Knowledge functional blocks.

These high-level objectives need to be further propagated to the targeted elements (AFs or MF core components), which requires their prior transformation into lower level policies. This translation process is accomplished by the Policy Derivation and Management function. After the translation process, low-level policies will be generated that correspond to:

- AF configuration options: These are policies sent to the AF to set its configuration and its behavior. The different configuration options are specified in the AF Manifest of each AF Class.
- Coordination configuration options can specify:
    - Sensitivity, as a value [0, 1]; this configuration option defines how sensitive Coordination can be when identifying potential conflict situations, and how to assign mechanisms. This sensitivity affects the loop search algorithm in the identification of conflicts using conflict maps (see clause 4.6 about problems to solve), when the strength of a loop is too weak, it is not considered as a conflict to be addressed.
    - Orientation, to be used by Coordination to prioritize/weigh AFs according to the operator's performance objectives. It can be set on a per-segment basis and requires that AFs also include their orientation in their manifests. For instance, if AF A states "Energy" as orientation and another AF B "Load Balancing" as orientation, and Governance in general states that "Energy">>"Load Balancing", then the priority assigned by Coordination to AF B will be lower than the one assigned to AF A.
    - Knowledge configuration options such as optimization goals to reduce the communication overhead, or the definition of a threshold for the detection of untrustworthy AFs.

Supervision

The previous clauses describe the operations of the Human-to-Network function that facilitates the communication of human network operators with the MF core, network elements, and AFs. Conversely, the Human-to-Network function should provide information about the status of the abovementioned elements. This information refers to the configuration, status, contexts and alerts of networks, services, AFs, and MF core. The Supervision operation provides the human operator with a clear picture of current network and system conditions. Based on that, a human operator may decide to use the other operations to modify the behaviour of the network or system, performing so called informed decision.

The process that decides which information is shown to address human operators' requirements is very important. The minimal amount of information that any MF-based Governance implementation should provide are: Alerts AFs may reach a situation where they cannot fulfil the specified goals in their current context. When all the self-healing attempts have failed, the AF should escalate the problem. In some circumstances, the misbehaviour of a set of coordinated AFs may be the origin of the problem, and therefore Coordination is informed in the first place. If all the feasibility checks of the coordination mechanisms fail, then the human network operator should be alerted.

The information attached to each alert should contain all the details needed for the human operator to take control over the situation:

- **Importance factor and priority:** an indication of the impact of this alert on the functioning of the network. Alerts should be prioritized, so those of the highest severity level should be processed first.
- **Originating system, Description, Category:** to better understand the nature of the alert.
- **Time markers:** history of the alert and its changes.
- **Related alerts and affected resources:** link to related alerts and tells affected entities.
- **Status:** indicates if the alert is active, closed or under investigation.
- **Root cause:** to be filled when the problem that caused the alert is diagnosed.

4.5.3.2 The Policy Derivation and Management Function

Operators need to control the behaviour of autonomic elements and to set technology-agnostic high-level objectives that need to be properly achieved by the different elements. More importantly, operators need to customize the operation of their highly complex, heterogeneous, and decentralized network on the fly. This necessitates to orchestrate various entities behaviour, which may belong to different segments and operational layers. Policies which will be used from Governance will span entities of different layers, administrative domains and network segments. The preservation of the relationships of policies between the respective different levels of abstraction, used and influenced by different entities/elements, requires policy continuum ([i.36]). Briefly, the purpose of the Policy Continuum is to provide a semantic linkage between different types of policies that exist at different levels of abstraction. Each of the levels is optimized for a different type of user that needs and/or uses different information. For example, a business user may need SLA information. That user is not interested in how the network is programmed to deliver QoS, just that the network is in fact delivering the right type of QoS to the right people. Conversely, the network administrator may want to "translate" the QoS that is implied by different SLAs into sets of different CLI commands to program the appropriate devices. This is a completely different representation of the same policy.

Policy continuum enables policies with a high level of abstraction, written from the concepts, terminology and structure/syntax (namely, different policy model and policy language) of this level, to be transformed to policies of a lower level of abstraction, with different corresponding characteristics. Policy continuum considers a policy as a potential continuum of many different policies with dependencies among them.

The proposed policy levels are the Business level, the Service level, and the AF level. The Business level corresponds to "Market, product & customer" layer of the highest conceptual view of eTOM (eTOM business process framework - Level 0 processes, [i.37]). The Service level and AF level corresponds to the "Service" layer and the "Resource (Application, computing and network)" layer, respectively. More precisely, the Business level policies are related to Strategy, Infrastructure and Product and Operations processes (eTOM business process framework - Level 1 processes). The Service level policies are related to Service management and operations processes of Operations, and the AF level policies are related to Resource management and operations processes of Operations. This policy hierarchy enables the definition of new sub-levels in the future, based on the emerging network and service requirements. Furthermore, the specification of policy levels based on eTOM, which entails the usage of basic common terminology, facilitates the policy translation between the different levels, enabling policy continuum.

The Business level policies, which are administration-oriented and technology-independent policies based on the technology-agnostic business objectives and service requests, are transformed into service policies that are service-oriented and technology-independent, and correlate service characteristics to specific network parameters. The service level policies, in sequence, are transformed into AF policies, which are technology-specific configuration commands related to the required operation of resources in specific network segments. The AF policies that are imposed on the corresponding AFs trigger specific actions from AFs, leading to vendor-specific commands executed by specific managed network elements. Concisely, the main objective of Governance operation is the automatic transformation of business goals into AF's configuration.

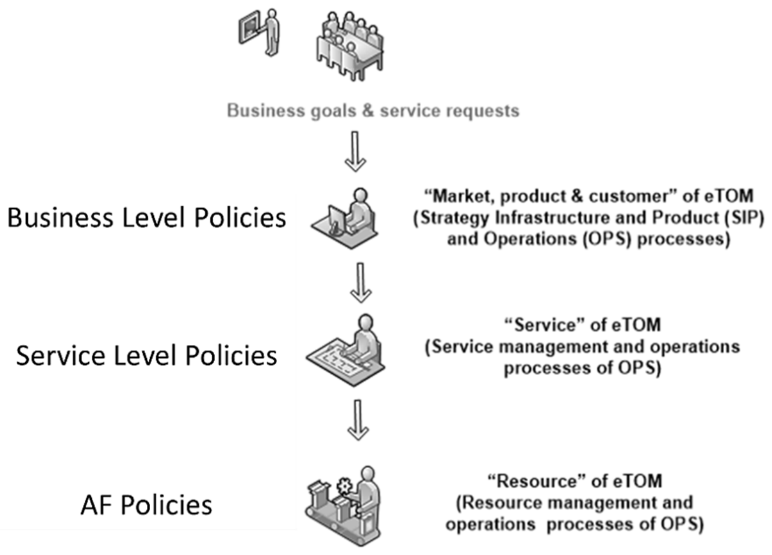
Business goals & service requests
Business Level Policies
"Market, product & customer" of eTOM (Strategy Infrastructure and Product (SIP) and Operations (OPS) processes)
Service Level Policies
"Service" of eTOM (Service management and operations processes of OPS)
AF Policies
"Resource" of eTOM (Resource management and operations processes of OPS)

Figure 18: Schematic representation of the business, service and AF policy levels

The Policy Derivation and Management function has been decomposed into the following operations:

- Build Business Policy.
- Policy Repository Management: Create/Update/Retrieve/Delete Policy operations.
- Validate Policy.
- Detect Policy Conflicts.
- Resolve Policy Conflicts.
- Translate Policy.
- Check Feasibility & Optimize.
- Policy Efficiency.

These operations can be applied at each of the levels of the Policy Continuum, except for Build Business Policy operation that only applies to the business level of the continuum.

4.5.3.3 The AF Management Function

The AF Management function collects and stores in the AF Registry all the management information of the deployed AFs. It also manages the state transition (including the activation and deactivation of the autonomic control loops) of the AFs and defines the reporting strategy that meets the operator needs.

This gives the human network operator the possibility of governing the autonomic behaviour of the network through the control of the operation of a particular AF or set of AFs. The AF Management function can be decomposed into the following operations:

- **AF Registry Management operations:** The information related to each AF instance should be kept in an AF registry, and therefore operations need to exist for its management: creation, update, deletion and retrieval of AF information.
- **AF Lifecycle Management operations:** An AF from the moment it is installed until the moment it is uninstalled follows its lifecycle.
- **Reporting Strategy-related operations:** AF reporting strategy operations allow the human operator to define the frequency and level of detail of the information, which the AF will report to the MF core. This operation may be useful under specific circumstances when the behavior of an AF is closely observed (e.g. when a malfunctioning has been detected, or when an event has been programmed that will increase the demand of services in a given area). In that case, the human network operator may wish to increase the frequency and the level of detail of the reported information.
- **Request specific behavior for AFs:** Request specific behavior for AFs is an operation which permits Governance to request specific control of AFs, in order to modify the mode of operation of a given AF or group of AFs. This feature allows for instance the network operator to set AFs into the so-called "under trial" mode, where AFs are not allowed to enforce actions on the network. The behavior of the AF can then be observed without risking the network operation.

There are two immediate usages for the "under trial" mode: observation of a misbehaving AF and certification activities. In the former case, when the online trust mechanisms find a deviation of the runtime behaviour of an AF with respect to the specified behaviour, the human network operator may decide, instead of stopping the AF, the deactivation of any kind of actions of the AF on the network and still observe its activity. Alternatively, instead of manually requesting the modification of the AF behaviour, the human operator may use the Human-to-Network function to set thresholds, when certain behaviours of the AF need to be (de-)activated. When measured trust for a given AF drops below the threshold, the trust mechanism in the Knowledge block issues a notification to the Governance, which triggers the AF Management function to set the AF in the "under trial" state. Similar observational needs may appear when a new AF is deployed during a test process or at the early stages of deployment in production environments, allowing the examination of the AF activity and its process of decision making without allowing the actual execution of corrective or preventive actions on the network elements.

Given an AF in operational state runs under the control of the Coordination functional block, the communication between both blocks is fundamental in this case, so Coordination can take appropriate actions to guarantee the stability of the network. Take for instance the case when the AF to be set "under trial" is part of a group of AFs being orchestrated. Given that one of them will be set to the "under trial" state may affect the behaviour of the group and therefore Coordination may need to send new control policies to those AFs. To ensure the proper coordination of the two blocks, Governance will not communicate directly with the AFs, but will request Coordination to control a given AF. This request will be in the form of policies, which will be realized by the Enforcement function. As part of these policies, Governance may request different options to Coordination. It may request a change in Coordination mechanism settings, to see the behaviour of the AF as if it was acting and to assess the influence of this AF on the other AFs, or it may just wish to observe the AF working in isolation from the other AFs.

4.5.3.4 The Enforcement Function

Enforcement encapsulates the communication mechanism between Governance and AFs or other MF core components. It allows the other functions of the Governance block to be independent of the communication aspects for the interconnection with AFs and core blocks. In the case of AFs, the communication is mainly achieved through the Mandate object.

4.5.3.5 Examples of Governance Mechanisms

4.5.3.5.1 Translation Mechanisms

The proposed policy translation method is based on a three-fold approach to maximize automation, retain low complexity and high precision, while being highly reusable. Firstly, the operator's High-Level Objectives are classified into categories (e.g. QoE (Quality of Experience), Energy Efficiency), while a set of KPIs are assigned to each category. Secondly, the translation process is not realized from scratch. In the proposed approach described in this clause, the translation is led by a set of guidelines which are documented in a set of policy templates stored in a policy template pool. These templates have the form of XML documents with a predefined XML schema.

In the considered example, a set of high level objectives parameters (e.g. User Class, Availability) comprise the values of the predefined "Policy Variable" and "Policy Value" elements. Thirdly, the modelling of high level objectives and translation process is realized using ontologies (OWL, Web Ontology Language as detailed by Antoniou and Van Harmelen (2009) [i.38]) and ontological rules. Ontologies constitute a means to capture information and organize information and knowledge representation in a reusable and machine-readable format. Consequently, ontologies enable the representation and communication of business, network and AF information, and the development of reasoning schemes. To this effect, the utilization of ontology-based policy translation is suitable for the relevant MF mechanisms. The formulation and realization of the ontology should be suitable to support taxonomy among the defined concepts and specification of relations in the form of Subject-Predicate-Object clauses. Furthermore, the designed ontology should ensure consistency of the included captured concepts and inferring of relations by assignment of meta-properties to existing properties, enabling deductive and inductive reasoning.

The translation process adopts the Policy Continuum approach as described in the previous claus and is accomplished at every policy level from the Policy Derivation and Management Function, through mapping of the policies' parameters to information parameters (and respective attributes) of the ontology. According to this approach, a set of three different levels / views are defined (Business Level, Service Level and AF Level), each of which constitutes a different representation of the initial business goal. The policies of all levels are described in OWL. The policies of the business level are modelled based on the ontology reflecting the business level, close to natural language, while the policies of other levels are modelled on the policy language that relies upon the SID (see [i.39]) information model standardized. The ontologies of different levels are linked in OWL by means of interoperability relationships between classes, which express the interrelation between subsequent levels, while SWRL (Semantic Web Rule Language) rules are used for the translation of the policies. The translation process is illustrated in Figure 19.

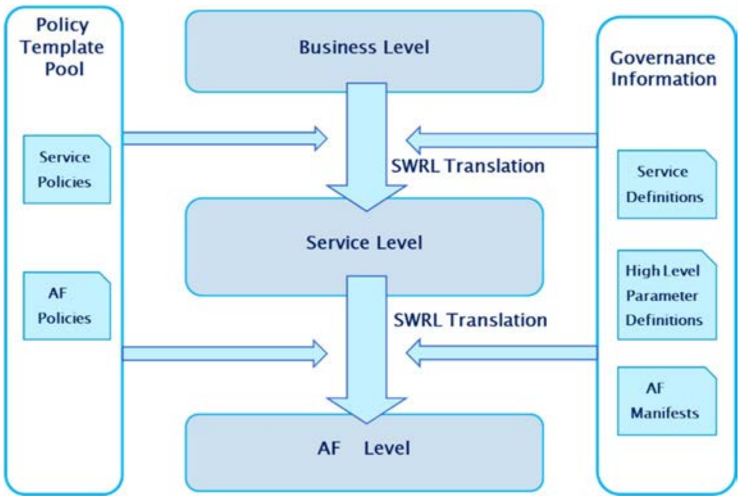

Figure 19: Policy Translation

The translation process comprises the following steps:

- Step 1: The initial Policy Variables which constitute High Level Objectives for the operator, are classified into High Level Objectives' Categories (HLO Category) based on operator's selections. For instance, HLO "Availability" may be comprised of four categories (e.g. Excellent, Good, Normal, Critical). Through the use of Human-to-Network function, the operator has the possibility to introduce both High Level Objectives and their categories respectively.

- Step 2: Based on High Level Objectives Definition, Build Business Policy operation expresses them in the form of business level policies.
- Step 3: The initial translation from business policies to service policies is realized based on High Level Objectives' Category to KPI mapping, the selected combination of high level semantics and the mapping of available services to KPI values (e.g. KPIDelay < 50 msec). For each HLO Category a series of service policy templates are extracted from the policy template pool. The selection of the appropriate policy template is done based on the set of KPIs involved and the initial classification. Each policy template is a policy skeleton which contains the policy structure and the policy variables, while the values of variables are missing. During the translation process the missing values are filled and the instances of policies are generated.
- Step 4: On the second level of translation each service level policy is further translated into a set of operational AF level policies. The translation is realized by using a set of AF templates, including KPI/parameter-related information filling them with KPI/parameter's values. In this step a mapping is realized between the involved KPIs/parameters and the available operations, inputs and outputs of the available AFs, which are described into the AF Manifest.

In its general form an AF policy is formed and described based on the SID (see [i.39]) policy model. Examples of some indicative policies are:

- Energy Efficient Policy: IF (time > 3:00 AND time< 8:00 AND NetworkUtilisation < 0,2 AND PercentageOfInactiveENodeBs < 0,1) THEN SET PercentageOfInactiveENodeBs = 0,1.

The AF policy implies that the AF that will receive the policy from Governance has the capabilities to monitor the network utilization (KPI) and control the percentage of inactive eNodeBs in the network:

- QoS/QoE Policy: ON SLAViolationAlarm IF SLAClass = Gold AND AdmissionControlThreshold > 0.9 THEN SET AdmissionControlThreshold = 0,9.

The above AF policy implies the specific AF can receive SLA violation alarms and control the threshold of admission control mechanisms of the access network.

4.5.3.5.2 Semantic-based Approach for Policy Conflict Detection

The semantic-based approach for the conflict identification described concerns the identification of parameters and their assigned values in the event, conditions and action clauses that can lead to a conflict between policies. This implies the definition of the suitable classes, properties and axioms. A relevant taxonomy between concepts defined for conflict identification in OWL is illustrated in the next figure. For example, an information piece can be correlated to:

- Parameter (observable and adjustable) comprising ComparableParameters (where predicates like isLessThan/isGreaterThan can be applied to their values) and NonComparableParameters (where there exists no natural ordering among their values, but isEqualTo/isNotEqualTo can still be applied);
- Value/ValueSet, comprising ComparableValueSet, (which has a lower and upper bound, either exclusive or inclusive) and NonComparableValueSet (which is a set of values, a union, or an interclause of sets); and
- ParameterType (which is the data type to be used for assigning actual values).

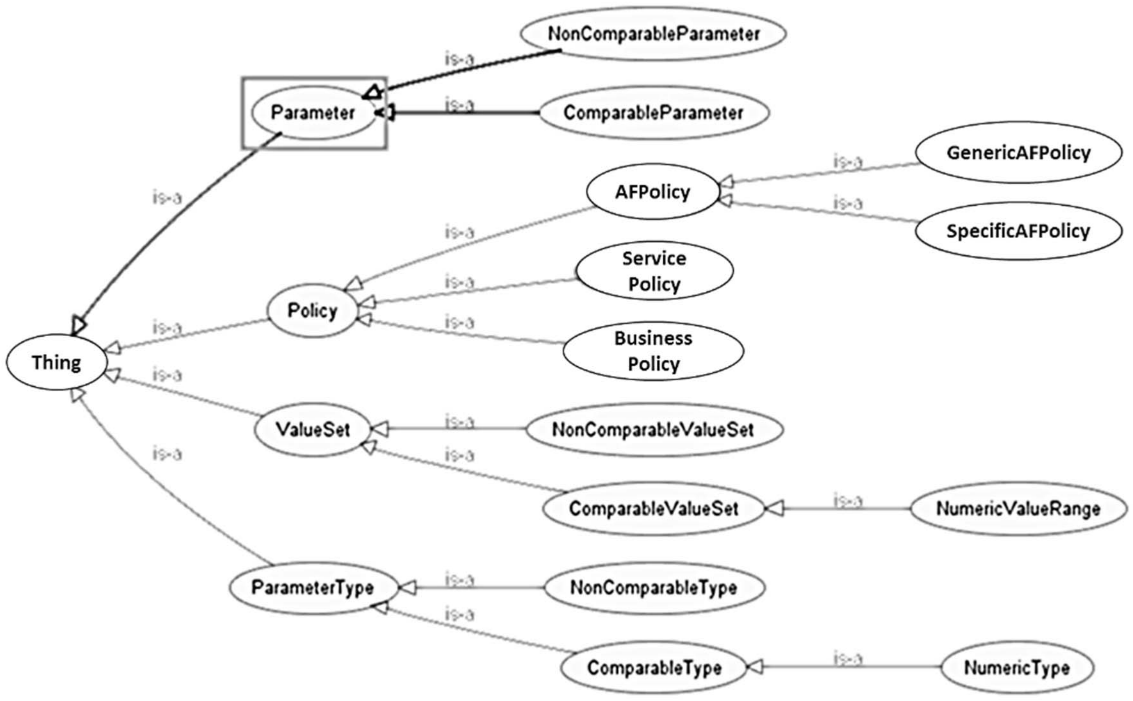

Figure 20: Taxonomy between concepts defined for conflict identification

In the case of the action clauses, the identification of possible conflicts is transformed into the detection of the alteration of the targeted parameters from the policies. Then the focus is on the action of the active policies, namely the targeted control of specific parameters. For example:

- policyA minimizes parameterX (Goal of policyA);
- policyB maximizes (or maintains value of) parameterX (Goal of policyB).

These policies are identified as conflicted policies.

4.5.4 Implications

Building a governance framework implies architectural and design considerations in at least the five main functional areas previously identified: business language, translation, reasoning, policies, and configuration enforcement. The network governance functionality should be developed to provide the operator with an appropriate, human-friendly governance interface, simple enough to be used by non-highly specialized technicians, and offer mechanisms that assist the operator to express goals, objectives, constraints, and rules to ensure the desired operation of its autonomic network.

Its components can be structured in four main functions (see Figure 17) which provide means to:

- Guide an operator (or customer) to express a business intent.
- Translate or decompose the business intent into service intents, coordinating governance policies with collected knowledge.
- Translate the service intent into concrete AF operations.
- Enforce communication between the Governance function and AFs and with the Coordination and Knowledge components.

4.5.5 Proof of concept

Network Governance has been validated by multiple evaluations in projects, scientific publications and demonstrations, such as:

- EU funded FP7 UNIVERSELF project (see also [i.31]).
- EuCNC (ex-FUNEMS) conference, Unified Management for Converged Networks, 2013 (see also [i.32]).
- A concrete implementation of Governance automation is also shown in clause 4.4.

4.5.6 Relevance for ZSM

Governance is an essential means to automate and simplify service and network management, but presents also its own design challenges. Such a governance functionality should be further investigated in the scope of the ZSM Architectural Framework with new or extended management services. In particular, the governance functionality should be connected to how the composing parts of the system interact with each other: they discover capabilities, requirements and constraints supported by each, eventually negotiate or dynamically trade responsibilities and implications (e.g. areas of application, separation of concerns), define conditions of their interactions (escalation, delegation and coordination) to support the targeted autonomy level. Most of the negotiation can be set at the instantiation time, but it should be possible to change the negotiated parameters over the life of the system, e.g. through proper governance interfaces, operations and lifecycle management.

There is also certainly a link between the governance functionality and the management of closed loop; for instance, to set their goals and operating regions and policies, their levels of autonomy and supervision. Further investigation in the relevant work items is encouraged.

4.6 Means of Automation for Network Stability

4.6.1 Motivation

Within the network, autonomic functions (AF) are likely to compete with one another, either to control same network parameters or to influence different metrics. This can result in conflicts and instabilities.

The role of coordination when managing multiple AFs is primarily to protect the network from instabilities and effects due to the presence of multiple AFs running concurrently. It ensures the proper triggering sequence of AFs and guarantees their stable operation. To this end, the coordination function defines conditions under which AFs will be executing (i.e. produce their output), considering operator service and network requirements.

4.6.2 Problems to be solved

Coordination aims at steering the network towards a better "operating" point, by avoiding or mitigating detrimental interactions between AFs that may lead to unstable and oscillatory behaviours, as illustrated in the prototype demonstrated by Koutsouris et al. [i.41].

The first step of such a process is the identification of these interactions and their classification to determine which ones need to be handled, i.e. conflicting interactions.

The second step clusters the identified interactions into groups that can be handled together while insuring the proper behaviour of the network.

The third step is the instantiation of coordination mechanisms well suited to handle each group of interactions previously identified.

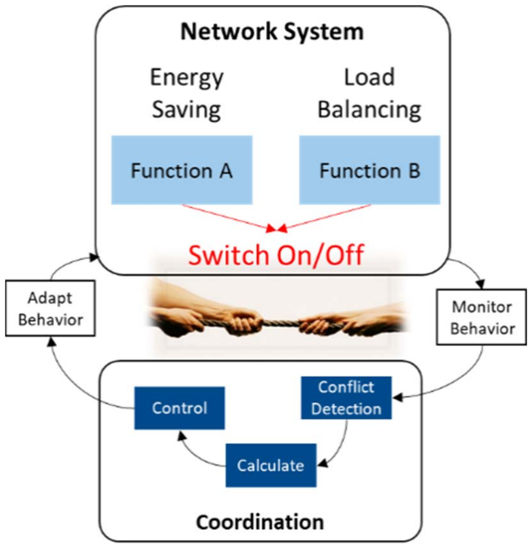

Figure 21: Three steps of coordination to achieve systems' stability: detect, calculate, control

Potential Conflicts

The mathematical problem to solve is computationally hard and complex in some matters. Computationally hard, because the number of interactions between AFs is big. It may become complex, because the nature of these interactions cannot always be predicted, and some knowledge may be lacking to identify all of them.

Radio Access Network (RAN) is one of the networking environments where autonomic functions have been deployed first (based upon the 3GPP Self Organizing Network - SON functions specification, see [i.40]). Taking time to analyze the SON functions, looking at which are the classes of parameters modified by each of those, the classes of metric used as input (also named KPI in the RAN context), and when trying to build the graph of these relations, results in a highly meshed graph. This is the static conflict map drawn in the below (see [i.42]).

From this interaction graph, one can deduce that SON function 4 is Mobility Robustness Optimization (MRO), which modifies the Handover selection from monitored inputs which are Handover errors, Number of Handovers and Ping-pong handovers, respectively.

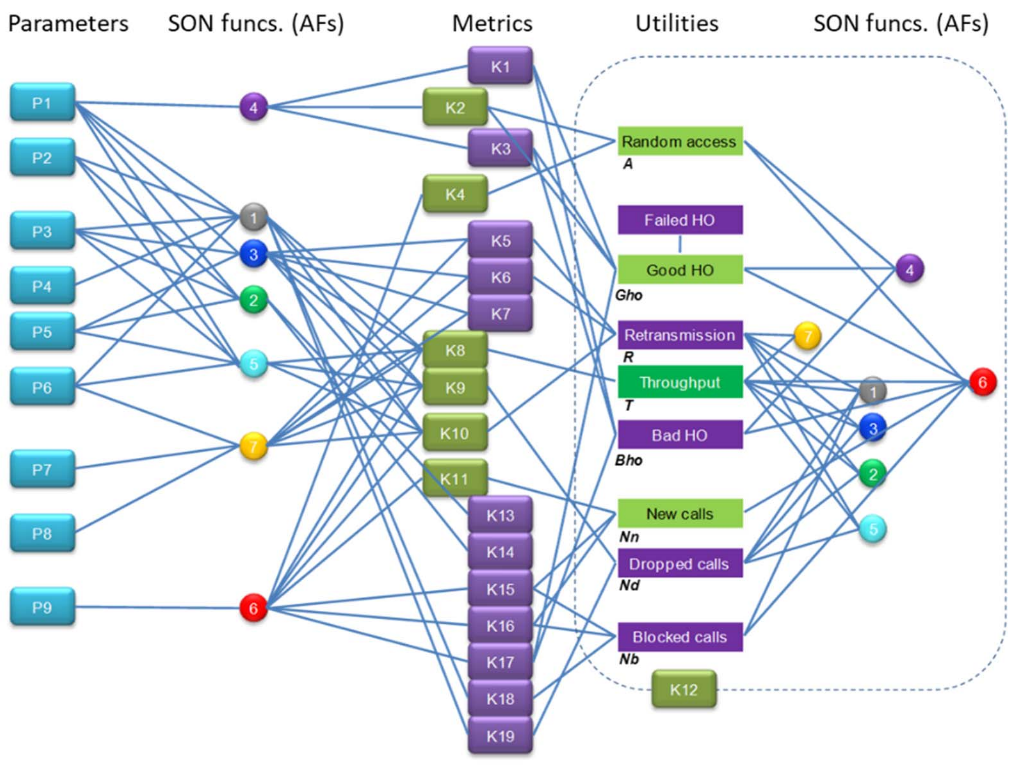

Figure 22: Static conflicts map derived from 3GPP SONs definition

This map shows for the controls (SON functions), which are the classes of parameters modified and which are the classes of metrics monitored, and finally which is the efficiency they aim at optimizing.

This static map already looks complicated, though it only displays static interactions. In the static map, the graph vertices are AF classes, the 'MetricClassDescriptor', and the 'ParameterClassDescriptor'. A more accurate view of conflicts would consist in plotting the deployed graph with instantiated vertices (AF instance, 'MetricDescriptor' and 'ParameterDescriptor'). Taking the example of the MRO AF, when deployed over multiple eNodeBs, there would be a vertex of Handover error per eNodeB (respectively other metrics). A deployed conflict map means that:

- For an AF that is deployed over multiple resources, the number of actual metrics monitored is multiplied by the number of resources hence multiplying the number of metric and parameters vertices and the number of edges between those.
- If the same AF Class is deployed multiple times (with multiple instances), each instance responsible for resources that are neighbours of resources of other instances, this will also multiply the number of atomic graphs interrelated by edges materializing the coupling between neighbouring resources (or more exactly between vertices of neighbouring resources).
- The graph also contains edges representing the coupling between different instances of the same metric class (the same parameter of neighbouring resources) or even between instances of metrics having any other type of relation/dependency.
- Finally, the graph also contains edges representing influences between instances of parameters and instances of metrics (i.e. a parameter whose value influences the metric of another AF).

When the graph is plotted with the entire set of dependencies for the whole network, then the network operator ends-up with a very complicated graph. Additionally, deriving the coupling in-between metrics on one hand and the influences between parameters and metrics on the other hand is a complex task. It either requires an a priori extensive knowledge of the network and of its topology plus knowing the relations between metrics, but still there is a risk that the graph may not be exhaustive or, even worse, error-free.

Types of Conflicts

Conflicts between AFs come from various types of interactions between these autonomic functions.

The most obvious conflict appears when two competing AFs decide over the value of a same parameter. Of course, this conflict can occur only if the parameter in-between is the same, meaning the same instance of parameter, not limited to the class. An example can be made with a competition between a congestion avoidance AF and an energy efficiency AF, if they both manipulate the metrics of router links, they would obviously compete only when both AFs are deployed over the same router links. This kind of conflict appears in the conflict map as shown in Figure 23. Figure 23 displays two autonomic functions and the graph shows the conflict and the closed control loops of each AF.

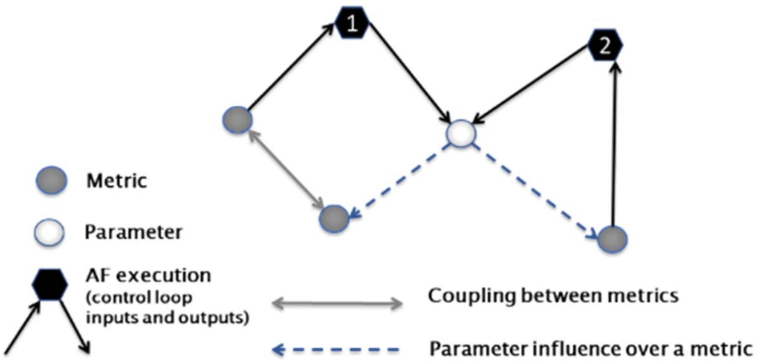
1
2
Metric
Parameter
AF execution
(control loop inputs and outputs)
Coupling between metrics
Parameter influence over a metric

Figure 23: Conflict between two autonomic functions modifying the value of the same parameter

Another type of conflict occurs when two AFs rely on metrics which are coupled the one with another (or have some degree of coupling or are the same) (shown in Figure 24). Each AF will change network parameters to augment the value of their utility function. The risk appears when the relation between utility function of each AF has variation opposite the other one regarding the coupled metrics (expressed mathematically: the partial derivative utility function of AF#1 has an opposite sign as the one of AF#2).

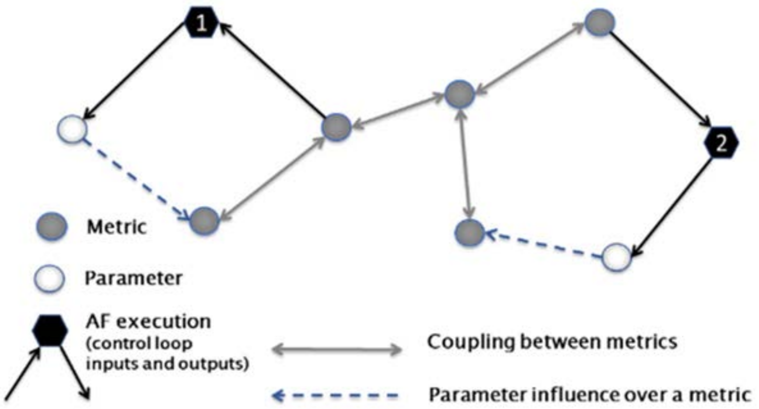
1
2
Metric
Parameter
AF execution
(control loop inputs and outputs)
Coupling between metrics
Parameter influence over a metric

Figure 24: Conflict between two autonomic functions using coupled metrics as input

Another type of conflict occurs when multiple AFs create a control loop that they close all together, as shown in the below figure. Figure 25 shows both the control loops of each individual AF and the broader control loop achieved by their interaction. Depending on the derivative of their utilities, this may lead to oscillation.

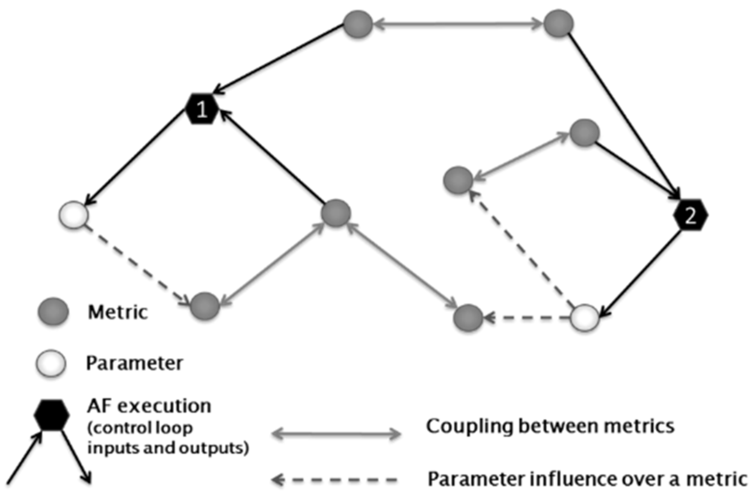
1
2
Metric
Parameter
AF execution
(control loop inputs and outputs)
Coupling between metrics
Parameter influence over a metric

Figure 25: Conflict between two autonomic functions forming a closed control loop

The types of conflict presented above can occur with more than two AFs (e.g. an interaction can involve three or more AFs), which makes the problem of identification even more complicated.

4.6.3 Solution Principles and Concepts

4.6.3.1 Times of the Identification of Interactions between AFs

The Coordination function manages AFs; hence its operation is related to the different states of AFs namely during their installation phase, their instantiation phase and when being in the operational state. Hence, the coordination function also presents a lifecycle consisting of these 3 different states, in which the coordination function behaves according to the following descriptions.

During the installation phase, a common description of the autonomic function attributes (metrics, parameters, actions, capabilities, etc.) leads to the construction of a "static interaction map" from the a priori knowledge that can be derived/inferred from the AFs control loop relationship. The static interaction map can be used as a first element by the operator (or mechanism) to (pre-)define policies and priorities as coordination strategies to manage the a priori conflicts that have been identified. There is an example of such an analysis focused on 3GPP Self Organizing functions (see Figure 22).

During the instantiation phase, autonomic functions are sequentially deployed then registered but are not acting on the network and its resources yet. At this stage, for each instance of AF and on a per resource basis, coordination is building an inventory of the metrics monitored, and of the actions performed. Augmenting this inventory with the relations between these metrics and these actions and their relationships can be realized, resulting in a "dynamic interaction map". The dynamic interaction map provides the basis to identify conflicts that will happen at run-time, categorize them and plan for the appropriate coordination strategies/mechanisms.

At runtime, conflicts may happen, and arbitration is driven by the coordination strategies and available mechanisms. This is also the stage where new dependencies can be observed and inferred, ultimately resulting in updating the dynamic interaction map and possible adaptation of the coordination strategies and mechanisms.

Identifying Conflicts During Instantiation

This type of conflict identification is based on a priori knowledge of conflicting parameters that is captured by the Management Framework system (mostly offline) and is then exploited to identify conflicts at runtime. This type of conflict identification is triggered upon configuration changes in the network or in the MF (e.g. the assignment of a new mandate to an AF), to detect and prevent conflicting setups.

The main advantage of such identification is that it detects conflicts proactively (i.e. before they occur). The major disadvantage on the other hand, is that networking experts' knowledge need to be carefully passed to the system through some manual, human-involving way, which is most often error-prone and time-consuming. In this respect, the Management Framework employs knowledge engineering techniques such as ontology, logic programming, and reasoning languages to empower static conflict identification and relax the requirements of a priori given knowledge.

The process of conflict detection contains multiple steps:

- Collect the instances manifest of all AFs and build the conflict map. The instance's description lists for each AF the parameters they modify and the metrics they take as input. Then the graph contains these vertices and the edges provided by AF (the black ones as plotted in the above different conflict type figures).

- Complete the conflict map, by coupling both in-between metrics and influences of parameter values over metrics (the other kind of edges plotted in Figure 25). To do so, the Management Framework employs ontology describing relations in between metrics and influences of parameters and a reasoning engine to derive the graph edges. Actually, it is simpler to have an ontology describing such relations in a static way, and expand the graph when mapping it with network topology information (e.g. "1" (when the ontology tells that the link metric value influences the load of the link, it is easy to apply this on every link of the network) or "2" (when the ontology tells the load of a link is coupled with the load of adjacent links, using a topology information easily allows plotting such coupling in the instantiated conflict map)).

- Parse the conflict map to identify loops involving more than one AF, and pinpoint those as potential conflicts.

This implies quite a sufficient knowledge injected inside the ontology, as the more accurate the ontology and topology information, the more accurate the instantiated conflict map. Additionally, the ontology and the reasoning mechanisms have both to be able to infer additional knowledge (e.g. the facts that "parameter A affects metric B", and "metric B is coupled to metric C" constitute sufficient knowledge to infer that "parameter A affects metric C", provided that the semantics of "affects" and "is coupled" are properly captured during design time).

Identifying Conflicts During Runtime

The role of the dynamic conflict identification is to "catch" conflicts and problematic situations that may have been missed during the previous identification phase. To do so, the dynamic conflict identification method needs to collect regularly AFs metrics as a function of AF parameters.

As such, the AFs will regularly share both their metrics and the values set to parameters. From metrics and parameter values, Coordination can calculate the matrix A with elements Aij, where Aij is metric i as a function of parameter j. This means that variation of metrics against all parameters are to be calculated. Once this is done the eigenvalues of matrix A are calculated. If all eigenvalues are negative then the set of AFs can be considered as stable; otherwise there is risk of instability and as such, the runtime conflict identification will finely monitor the AF behaviour to detect oscillations or untrusted behaviours, or even raise alarms.

Specifically, groups of AFs coordinated by a same instance of a coordination mechanism, can benefit from a dedicated runtime identification, as coordination can infer from that, whether AFs can keep operating in their current mode (and even relax gradually their constraints) or a new grouping under the available conflict managing mechanisms is needed.

4.6.3.2 Algorithms to Insure Coordination

The solutions to avoid conflicts between multiple control loops can rely on different categories of algorithms as detailed in [i.43] or [i.44] or [i.45]. In the context of a framework capable of handling any sort of autonomic functions not known a priori, algorithms specifically designed for a given interaction between control loop are not to be considered. The algorithms requiring little to no tailoring are better suited for the proposed framework. Hereafter is provided an overview of algorithm families capable of achieving coordination between autonomic functions, or at least to avoid conflicts:

- Random token: Such algorithms insure that each autonomic function is executing its control-loop one after the other, the sequence is following a random pattern.

- Hierarchical coordination: Such algorithms insure that each autonomic function is executing its control loop at different rates, e.g. for two functions: one is running fast enough to have time to converge in between two iterations of the slowest one (this algorithm requires proper settings with regards of the autonomic functions to coordinate).

- Efficiency bids: In such algorithms, each autonomic function predicts which improvement its executing of its control loop would bring, and communicates this bid to the coordination algorithm, which then picks the autonomic function promising the "best" improvement and grants it the right to execute. Then this process repeats itself.

- Multi-objective optimization: For such algorithms, the autonomic functions delegate the computation of their outcome to the coordination algorithm. They supposedly have communicated in advance their analytical utility to the coordination algorithm, which itself computes a multi-objective optimum, and returns to the autonomic function the output the autonomic function would apply.

A Generic Mechanism of Random Token

This method for managing conflicts belongs to the separation in time strategies and is the simplest possible mechanism. It sets the baseline for all other conflict managing mechanisms. It can be viewed as the last resort when other mechanisms cannot be applied, due to AFs being unable to provide the needed information required by mechanisms offering better coordination.

In such cases, AFs identified as possibly conflicting are waiting for a token before executing their autonomic loop. During the initialization phase of this mechanism, coordination will instruct AFs that they should wait for the token to run. During runtime, upon receiving the token, the AF will run its optimization cycle only once and, once finished, it returns it to coordination.

A Generic Mechanism Based on Hierarchical Coordination

This method for managing conflicts is based on AF control loops that operate on different time scales according to the dynamics of the resources which they control. By enforcing different time scales at these AFs, one defines a hierarchical control system which is denoted as hierarchical optimization. After each control action (e.g. parameter modification) of a slow AF, faster AFs act rapidly and converge. Hence the slow AF is seen as quasi-static by the fast AFs. If the periodicity of the fast AF is denoted as Tf and that of the slow AF as Ts, then convergence of the hierarchical system is guaranteed if, or in practice, a small value. In Game Theory, the slow AF can be considered as a "leader", while the fast AF is considered as a "follower"; and the optimal point of operation is known as Stackelberg equilibrium.

For this method to be applicable, the AFs will be able to operate periodically. This mechanism does not require any runtime information from the network or from AFs themselves, it only requires that during initialization phase, coordination imposes to AFs the periodicities they should follow. From this time on, the AFs are in principle allowed to operate without any additional constraints.

A Generic Mechanism Based on Efficiency Bids

This method for conflict avoidance can be regarded as an extension of the random token mechanism, with the key following differences:

- AFs need to be able to predict the outcome of their actions (predicted utility).
- AFs need to have the same periodicity, or at least periodicity equals to multiples of the same base periodicity.
- AFs -ideally- should be able to tolerate the temporary suspension of their actions and still converge.

In such cases, AFs with the above characteristics can be coordinated by an efficiency bids mechanism. This mechanism does require the reception of predicted utilities from the controlled AFs. AFs report their predicted utilities only when their triggering conditions are met. Then the AF with the highest predicted utility is selected to complete its control loop IF the triggering conditions are still valid. If not, then the AF with the second highest predicted utility is selected, etc.

During the initialization phase, this mechanism will instruct AFs that they will report their predicted utility when asked, and not to execute their control loop, apart when granted a token.

During runtime, coordination will be requesting from AFs their predicted utilities with a frequency defined by their base periodicity. AFs which meet their triggering conditions report the corresponding values and coordination will choose to which AF to pass the token.

A Generic Mechanism Based on Joint Optimization

This family of algorithms is the hardest to make it generic, and AFs can only be eligible to joint optimization at the cost of heavy design constraints. The principle is to solve a multi-objective optimization problem in a centralized way. Each AF was independently trying to optimize its own objective, whereas coordination will address a problem of optimizing N objectives which can be formulated as:

$$\min \vec{F}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_N(\vec{x})]^T$$

$$\text{subject to } g_m(\vec{x}) \leq 0, m = 1,2,\ldots.M \text{ where } \vec{x} = [x_1, x_2, \ldots, x_p]^T \in \Omega$$

where $\vec{F}$ represents the objective vector, g denotes the constraints and $\vec{x}$ is a *P*-dimensional vector representing the decision variables within a parameter space Ω. The mathematical problem can be solved by aggregating all the objectives into a single objective or optimizing one objective and treating the other as constraints. These methods produce a single solution as outcome, rather than a Pareto Optima.

During the initialization phase, coordination constructs the optimization objective and method that will be used to jointly optimize the set of AFs. Coordination also instructs AFs to disable their actuation (they'll only perform monitoring), and to report the monitored metrics when requested.

During runtime, upon request from coordination, AFs report the metrics they have monitored, then coordination proceeds to solve the multi-objective optimization problem and calculates the network parameter values that AFs will enforce on the controlled network elements. Finally, AFs enforce those values, which will remain valid until the next cycle of the centralized optimizer.

4.6.3.3 Coordination Mechanisms to Control AFs

This clause provides a synthetic view of the algorithms previously presented in an increasing efficiency order. The range of efficiency spans from minimizing the influence of conflicts (by avoiding concurrent actions) to jointly optimizing the utility of all AFs. Unfortunately, the increase of efficiency is gained at the cost of complexity growth both in setting parameters to the coordination algorithm and in terms of requirements on AFs API, as depicted in Table 3.

Table 3: Summary of features AFs need to support to be coordinated by each of the coordination algorithms

Algorithm name	Control the frequency of loop execution	Knowledge provided by AF to coordination	Suspending changes to parameters	Forcing values to parameters
Random token	X			
Hierarchical coordination	X	Average convergence time		
Efficiency bids	X	Predicted utility	x	
Joint optimization	X	Utility function	x	x

During the initialization phase, sets of AFs are assigned to instances of the coordination mechanisms. The selection and grouping of AFs under each instance depends primarily on identified conflicts, but it can additionally depend on factors such as proximity of AFs' controlled resources, policies coming from the static conflict identification. Finally, the choice of the coordination mechanism depends also on the AFs in the group, whether they are compliant with the chosen algorithm.

4.6.4 Implications

Reaching higher levels of network automation at local and global scales requires enabling higher degrees of autonomy for the mechanisms controlling and managing the network behaviours. Enabling such higher degrees of autonomy over distributed and sometimes independent systems while preserving the ability for the operator to govern the overall behaviour and stability of the networks involved, require in turn certain levels or forms of orchestration, choreography or coordination. Hereafter, this clause will refer only to coordination.

It remains to be described how such coordination schemes are designed and embedded in the ZSM management reference framework and what normative (or not) specifications are necessary, and what is matter of implementation.

Several important aspects should be considered for the design, specification or recommendation of the coordination functionality and mechanisms such as:

- Scalability: ability to coordinate simultaneously the functioning of hundreds or thousands of control loops.
- Distributivity: avoid single point of failure and centralization of the coordination function.
- Robustness: design and operate coordination schemes robust to changing conditions and use cases, while preserving levels of performance.
- Performance: e.g. speed of decision based on number of interactions to manage, amount of resources required to attain certain performance levels and trade-off with cost.
- Stability: short term vs. long term coordination, local vs. global stability e.g. approaching coordination beyond myopic vision (i.e. only short-term and local coordination), stability of decisions and operation regimes.
- Coping with uncertainties and priorities.
- Constraints on sharing and communication of information (registration, advertising/discovery, storage) and ability to identify and select the information relevant to achieve efficient coordination.
- Scope of control: clustering of coordination groups or domains.
- Dynamic negotiation of scope of control and re-clustering, inter-cluster negotiation for coordination optimization.
- Integration with intent-guidance for coordination targets.
- Security: TBD/FFS.

All these aspects have to be considered also relative to each other.

4.6.5 Proof of concept

The different mechanisms to ensure network stability have been validated in by multiple evaluations in projects, scientific publications and demonstrations, such as:

- EU funded FP7 UNIVERSELF project (see also [i.31]);
- EuCNC (ex-FUNEMS) conference, Unified Management for Converged Networks, 2013 (see also [i.32]);
- IEEE IM conference, Conflict free coordination of SON functions in a Unified Management Framework, 2013 (see also [i.41]).
- Workshop on Managing Autonomic Network Functions, Wednesday, November 20th - Orange Labs - Issy Les Moulineaux, France.

4.6.6 Relevance for ZSM

Providing a coordination functionality for ZSM represents a key enabler for network and service automation, in particular to increase the overall system stability and in support to the management of interaction and potential conflicts between closed loops. A common coordination functionality is therefore relevant and of importance to ZSM.

ZSM should investigate further how to enable the integration of such a coordination functionality and its mechanisms within the ZSM architectural framework, allowing a flexible and versatile use of the function.

4.7 Means of Automation: Reinforcement Learning

4.7.1 Motivation

In general, Machine Learning (ML) algorithms can be classified into three different categories:

i) supervised learning;

ii) unsupervised learning; and

iii) reinforcement learning.

In supervised learning algorithms, a set of labelled historical data is used to create a model to be used for deriving the outcome from new input data; an example of application in the telecom industry is the prediction of network load based on historical data and traffic features. In unsupervised learning, there is no need for labelled data, but instead, historical data is used to extract hidden patterns or structures to be applied to future input data; examples of applications in the telecom industry are anomaly detection or user grouping based on traffic profile. Finally, in Reinforcement Learning (RL) the goal is to learn a policy that will optimize the outcome of a process or system based on interactions of an agent with an environment that is unfeasible to be properly modelled.

In RL, an agent tries to learn an optimal policy to act upon an unknown environment based on the system states and a reward signals. This technique allows the development of self-learning agents that can interact with environments that are hard or impossible to be modelled and yet optimize the outcomes. RL-based systems require minimum human intervention or threshold-based rules/policies and can dynamically adapt to changes in the target environment.

The history of RL is very long and rich, including works concerning with learning by trial and error approaches and works concerning optimal control based on value function and dynamic programming. Most of the prior work date in the early 1980s, but recently the use of RL has resurged with many different applications. Many successful projects have used RL to solve resources management problems in general and have applied it to robotics [i.46], [i.47]. Some recognized advances also took place in the area of recommendation systems, advertising, and bidding [i.48]. However, the most famous and recent RL advances came from applications related to games, where intelligent software was able to play Atari^{TM} games [i.49] and even beat humans in Go^{TM} [i.50].

As telecom environments are very complex and rich in configuration parameters, it becomes a target for works that employ RL algorithms. Virtualisation technologies create new opportunities for network planning, provisioning, and assurance using RL, especially to realize self-learning, dynamic capabilities in the area of self-optimization, -configuration and -healing. RL is a promising technology for NW automation to build self-healing, parameter-free, zero-touch systems.

4.7.2 Problems to be solved

Applying RL in a telecom system incurs many challenges that need to be tackled for robust and trustworthy operation [i.51].

One first problem is the availability of data and data sample sparsity. As with any machine learning algorithm, reinforcement learning works based on historical data and previous experiences. Differently, from many other scenarios where simulations can be used to run thousands of test episodes, in telecom systems data samples are hard to be generated, and the exploration of the system may take a long time. Therefore, learning a policy based only on interactions with a real system may be impractical. One way forward is to employ a simulation-assisted learning process where a synthetic environment is used (optionally together with the real environment) for exploration [i.52]. That brings out the next challenge: telecom systems are hard to be modelled and simulated.

Telecom environments are very complex and involve systems with different requirements in terms of scope (e.g. resource optimization in the access network, transport network, cloud), time-scale execution (e.g. real-time analytics, optimizations in core network) and reliability (e.g. low-latency reliable applications, mobile broadband). The complexity of such systems translates into imperfect state information that prevents an efficient learning process and may cause undesirable outcomes. Beyond the issues caused by inaccurate modelling of the environment, telecom networks have very strict requirements on availability, stability, robustness, and efficiency that may hinder the use of RL without proper supervision.

There are different strategies to address these requirements, such as pre-training, transfer learning, shared learning, semi-supervised RL, and the use of simulation-in-the-loop. Some deployment options for RL agents in live production systems exist, options range from conservative static, controlled policy update to dynamic online learning and policy improvement in the live system.

4.7.3 Solution Principles and Concepts

The mechanics of an RL algorithm define how the following concepts interact:

i) environment;

ii) states;

iii) actions;

iv) reward signals;

vi) value functions; and

vii) policies.

The RL problem can be formalized as an MDP where an agent learns how to interact with the environment, as illustrated in Figure 26. An *environment* is a stochastic system that can be represented as a set S of possible states where the agent takes *actions* at each time step $t = 0, 1, ...$. These actions are selected from a set A of possible actions to be taken in a given environment. Based on the current state of the environment s_t and the action taken a_t, the transition probability distribution P determines what is the next state s_{t+1} to be observed by the agent, i.e. $s_{t+1} \sim p(s_t, a_t)$.

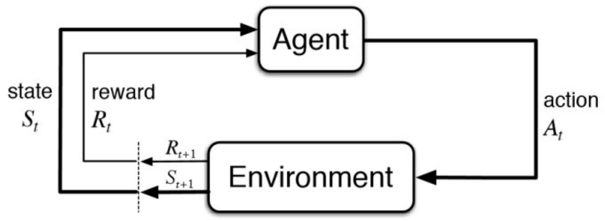

Figure 26: RL building blocks ([i.53], p.48)

In general, RL-based system can be modeled by the tuple (S, A, P, R), where S is the state space of the environment, A is the set of possible actions an agent can perform, $P_a(s_t, s_{t+1})$ is the transition probability distribution, and $R_a(s_t, s_{t+1})$ is the immediate reward signal.

The *reward signal* is received from the environment as a result of the state-transitions induced by actions. Such a signal tells how good or bad an action was when taken in the current state of the system. The objective of an agent is to maximize the total reward received in the long run. Therefore, the agent uses the reward signal as a way of learning how to improve its behavior in the future. Like policies, reward signals may also be stochastic functions of the states and the actions taken by the agent.

While the reward signal expresses immediate benefits, a *value function* expresses the return in the long run. Such a function provides an estimate of the quality (*value*) of a certain state by estimating the total reward an agent may expect to accumulate over time when starting from that state. It is not rare that, in a given state, certain actions leading to low immediate reward will lead to large long-term returns.

Even though both rewards and values express the return of actions, an agent is usually more concerned with the latter. An optimal agent will seek states that bring the most value (not rewards) because higher values will bring the highest return on the long run. The rewards are easily derived from the environment, while the values are estimated and re-estimated from the sequence of interactions the agent performs with the environment.

The *policy* defines the behavior of the learning agent when interacting with the environment. It is a mapping that tells the actions to be performed by the agent given the perceived state of the system. The final goal of an RL-based algorithm is to find the optimal policy that optimizes the return. Policies may be deterministic but also stochastic when they define a probability distribution of actions given the current state. An agent in RL is composed by an intelligent algorithm that derives the best policy from rewards and state observations. In a management framework such as ZSM, the agent can be realized by management services responsible for the decision-making processes.

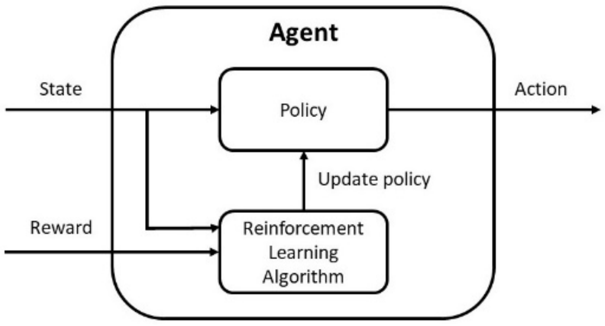

Figure 27: Agent in reinforcement learning algorithm

When the transition probability matrix is known, the RL agent can interact with the *model* of the environment directly, and a closed formulation of the RL problem can be solved. However, many real scenarios are too complex to be modeled or even unknown, which imposes the challenge of solving a model-free problem.

Model-free RL requires heuristics to determine the best policy. One commonly employed model-free algorithm for RL is Q-learning ([i.53], p. 131). Q-learning learns the policy to be followed by calculating the so-called quality function (Q-value, the reason why the technique is called Q-learning). The Q-value represents the expected reward accumulation based on the possible states and actions, in opposition to a reward value that only considers the current state.

In Q-learning, a Q-table (representing a function $Q: S \times A \rightarrow \mathbb{R}$) is initialized with arbitrary values. For each time step, the agent takes an action a_t, receives a reward r_t, and enters a new state s_{t+1}. Such an interaction with the environment, provides new data to update the entry $Q(s_t, a_t)$. The new $Q(s_t, a_t)$ value maintains part of its current value and adds a part that estimates the future return obtained from the state s_{t+1}, as in Equation (1):

$$Q(s_t, a_t) \leftarrow (1-\alpha)\, Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) \right) \quad (1)$$

The constant α is the learning rate that defines how strongly new experiences will affect the Q-values. The constant γ is the discount factor that defines how much the estimate of future return will influence the update of Q-values. Both values are in the $(0, 1]$ range, but it is common practice for α being close to zero and γ being close to one.

However, tabular approaches such as Q-learning can become impractical when applied to complex scenarios with large state-action spaces. A common approach to overcome such problem is to employ function approximators that can cope with both high dimensionality and high cardinality, e.g. Neural Networks (NN), to estimate the value functions and enable the learning of approximated policies. In cases where deep NN (i.e. neural networks with large number of hidden layers) are employed, the approach is called Deep Reinforcement Learning (DRL). More specifically, in the case of Q-learning, such combination led to the proposal of the Deep Q-Networks (DQN) [i.49]. Other well-known DRL algorithms include Asynchronous Advantage Actor-Critic (A3C) and Proximal Policy Optimization (PPO).

4.7.4 Implications

RL tries to find the best balance between exploitation and exploration. The agent derives a policy that exploits the knowledge gathered from interactions with the environment. However, as in any other trial and error process, there is a need to explore unknown states and try different action before a good policy can be derived. Moreover, as complex systems change over time the ability to explore an environment is not only needed for the convergence of the best policy but also desired to track changes in the environment and continuously optimize the policy. During the exploration phase, there is a higher risk of performance degradation and even the possibility of leading the system to unsafe states.

Interpretability and safety are important in any machine learning system and especially critical for RL. The exploration part of RL has to be carefully aided by protection mechanisms if applied to a mission-critical application. Such mechanisms may include experts´ feedback and ways of human intervention for risk management. Several advances in RL algorithms include research in topics such as multi-agent control, transfer learning, meta-learning, algorithm generalization, and more efficient training methods for sparse and real-time systems. In transfer learning, the policy is either trained in a simulator or a controlled system before it is put in the target environment. Transfer learning is one advance that has many implications for the success of RL in the telecom system. More details on this technique can be found in clause 4.8.

Many of the cutting-edge advances in RL are done by IT companies in the web and cloud systems. Such advances could also be applied to the telecom system and used as a basis for specific developments focused on network and services management.

4.7.5 Proof of concept

There are many examples of RL-based solutions in production, mainly from IT and cloud companies. Some of the recent advances are also publicly available through projects such as Gym from OpenAI, Horizon™ from Facebook™, and RAY™ from RISELab™ UC Berkeley. A number of proof of concepts have also been developed for telco companies and applied to communications and networking.

Self-learning agents have been applied in virtualised core (vMME) for VNF autoscaling using deep reinforcement learning which is improving system performance with 25 percent compared to predefined thresholds [i.54]. It has also been applied to self-heal signaling storms by adjusting throttling on vMME to mitigate the signaling storm.

Envisioned 5G systems with dense deployments, employing advanced beam-forming capabilities that are required to handle high levels of mobility efficiently are examples of scenarios suited for RL-based solutions. An improved handover strategy based on multi-armed contextual bandits have been evaluated and showed promising results when compared to existing methods [i.55]. Contextual bandits have also been applied to optimize radio link adaptation and studies have shown improved spectrum efficiency in some scenarios. A joint activity between Ericsson™ Research and UC Berkeley's RISELab has investigated the applicability of RL to find the optimal antenna tilt angle per LTE cell. In this case, a RL agent tunes macro LTE cells' antenna tilt adaptively to user hotspot by leveraging Ray RLlib's™ scalability.

4.7.6 Relevance for ZSM

As explained above, Reinforcement Learning is a powerful method for creating ways of learning from experience and optimizing a return function based on interactions with an unknown environment. Therefore, it can be a useful enabler for network and services automation within the ZSM framework architecture. RL technique provides a flexible mechanism for implementation of autonomous agents that can be applied to optimize different pieces of the end-to-end service offering and can work with different contexts, from local parameter configuration to global SLA assurance.

There are many advantages of applying RL in ZSM and, hence, there should be further investigations of ways to enable the integration of RL within the ZSM architectural framework. This can further take place as studies that analyse the applicability of RL in specific scenarios, and identification of parts of RL framework that could be standardized to better integrated into ZSM architecture and allow re-use of implementation and interoperability between vendors.

Clearly RL is one enabler to be exploited in closed loop automation, and it should be considered in work items that address this topic in ZSM. This could be done in two phases, first applying RL as a companion of OODA-based closed loops, where steps of the closed loops are realized by components of the RL framework. Secondly, applying RL as a replacement of conventional control loop mechanisms and as a way of future evolving closed loop automation toward artificial intelligence.

4.8 Means of Automation: Transfer Learning

4.8.1 Motivation

In self-organizing networks (SONs), widely studied use cases include coverage and capacity optimization, handover/mobility robustness optimization, cell outage detection and compensation, and interference mitigation. Usually, state-of-the-art solutions for SON use cases can be classified into two categories:

1) **Optimization based on approximated mathematical models of networks**, such as in [i.64] and [i.58], which, however, due to the increasing complexity of the next generation networks, usually leads to unrealistic assumptions and impossibility in deriving closed-form mathematical models that can approximate the real scenario for system-level optimization.

2) **Online learning and optimization with feedback measurements**, as in [i.57], where models/algorithms try to adapt to real scenarios based on data-driven searching approaches, but also require long time to converge due to the lack of a comprehensive knowledge of the system.

The disadvantages of both the model-based optimization and the conventional data-driven searching approaches are clear: the former cannot adapt to varying or even brand new, unknown network states, while the latter requires a large amount of data to obtain sufficient information for enabling a fast convergence of the optimization algorithm.

NOTE: The cost of the data in the latter case, including data collection, data storage, and data transfer, is also high in terms of various types of resource consumption, e.g. driving tests, data storage, and data transmission delay.

Therefore, a novel approach to facilitate and accelerate network self-planning and self-automation mechanisms is foreseen.

4.8.2 Problems to be solved

The growing trends for next generation networks, namely dense deployment, new radio access technology, and data analytics, rise the challenges of rapidly adapting to unknown scenarios and making fast network optimization decisions even when limited information is available.

In this context, the proposed Means of Automation aims at solving the following problem:

How to accelerate the network self-planning and self-optimization mechanisms and achieve the "best performance" as a reaction to changes, such as following a new deployment, a hardware or software update, and the application of new radio access technologies or new self-optimization functions, while keeping the cost for trial measurements and data transfer low?

4.8.3 Solution Principles and Concepts

4.8.3.1 The Deep Reinforcement Learning framework

The proposed self-transfer optimization method is based on transfer learning within the **Deep Reinforcement Learning (DRL)** framework to accelerate the network optimization decisions when experiencing changes or new deployments in network environments. The DRL networks consist of at least the convolutional layers and the fully connected layers (an example is given in Figure 1). The convolutional layers are used to capture the spatial and temporal network environment dependencies, while the fully connected layers are used to reduce the dimension to the desired output dimension. Moreover, the method exploits the "transferability" of the knowledge embedded in the pretrained convolutional neural networks. The DRL model is characterized by a set of model ***parameters*** (e.g. weight matrices, bias vectors) and ***hyperparameters*** (e.g. number of layers, number of units at each layer, type of the activation function).

The following two scenarios are considered.

[Scenario I] Transfer knowledge from a pretrained optimization model to a new optimization model in the same system.

[Scenario II] Transfer knowledge from a previously existing system to a new system.

NOTE 1: An "optimization model" is defined by the input variables, output parameters, and optimization objectives. To distinguish it from the learning model (e.g. neural networks defined by number of the layers, number of neurons in each layer, activation function, etc.), hereafter "optimization model" and "optimization function" are used interchangeably.

NOTE 2: A "system" can refer to any network element (e.g. base station, WiFi access point, pico/femto cell), or sub-network/network consisting of multiple network elements.

The knowledge is "transferred" by reusing either partial or complete pretrained parameters and hyperparameters (e.g. those characterizing the lower layers of DRL model). The layers to be transferred can be selected based on a ***similarity measure*** between the existing and the new system. To adapt the model to the new system, the model can first be modified (e.g. by adding layers or modifying the output layer), then the modified model or the newly inserted layers can be retrained. The advantage is that the train/retrain of the new/modified layers/model is possible with a small number of measurements collected from the new system (i.e. smaller than what would be needed without transfer learning).

Compared against the state-of-the-art network self-planning and optimization approaches, our approach has the following advantages:

1) **Data-based.** Compared to the physical model-based approach, it learns the optimization decision process using transferred knowledge and thus with a limited amount of collected data in the new system, without explicitly approximating the closed-form expression for physical mechanisms in a complex network system.

2) **Knowledge exploitation.** The conventional model-based approach requires well-defined network model as a priori knowledge, and the conventional stochastic data-driven approach needs to learn from scratch by following the same learning process for every new scenario, while our approach can exploit the extracted knowledge from previous systems.

3) **Fast learning.** The transfer learning approach adapts to the new system by partially retraining or fine-tuning the pretrained model, which leads to a faster learning process and better performance when system is subject to changes.

4) **Reduced data cost.** The following three types of the data cost are considered: data storage, data transfer, and data collection. Since the knowledge is transferred to the new system by transferring partial (or complete) parameters and hyperparameters for a subset of the pretrained model, no measurement data needs to be transferred. This reduces the cost for local data storage and data transfer between the network elements. Although some storage space and transmission resource are still requested for model transfer, the required storage space and transmission resource are still significantly less than those required for saving a whole dataset and training a complete model from scratch locally. Moreover, since the approach enables quick learning with limited training samples collected from the new system, it also decreases the cost for trial/drive test and data collection.

4.8.3.2 Self-Transfer Optimization Network

4.8.3.2.1 Deep Reinforcement Learning Framework

A DRL framework similar to the one proposed in [i.60] is considered, but modified in such a way that it learns the mapping between the network environment measurements (given as inputs) and the optimization actions/decisions (given as outputs), where the corresponding system performance metrics are mapped as rewards. The DRL model consists of at least convolutional layers and fully connected layers, where the convolutional layers are used to capture the temporal and spatial correlations of the network environment, while the fully connected layers are used for reducing the dimension to the required dimension of the output actions. Figure 28 gives an example of the DRL model with two convolutional layers and two fully connected layers.

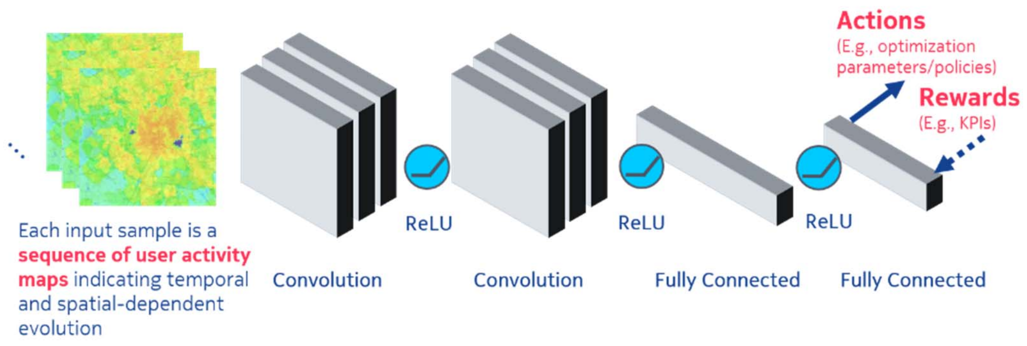
Actions
(E.g., optimization parameters/policies)
Rewards
(E.g., KPIs)
Each input sample is a sequence of user activity maps indicating temporal and spatial-dependent evolution
ReLU
ReLU
ReLU
Convolution
Convolution
Fully Connected
Fully Connected

Figure 28: An example of the DRL network

The DRL model reflects how the optimization agent interacts with the network environment through a sequence of actions and observed rewards. The goal of the agent is to select actions that maximize the cumulative expected reward. More specifically, a DRL network **approximate**s the optimal action-value function:

$$Q^*(\mathbf{s},\mathbf{a}) = \max_{\pi} \mathbb{E}\left[\sum_{n=0,1,\ldots} \gamma^n r_{t+n} \,|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}, \pi\right] \quad (1)$$

which is the maximum sum of **rewards** r_t discounted by γ at each time step t, achievable by a behavior policy $\pi = P(\mathbf{a}|\mathbf{s})$, after making a network **state** observation **s** and taking an **action a**.

The above optimal action-value function obeys the Bellman equation based on the intuition of recursion: if the optimal value $Q^*(\mathbf{s}', \mathbf{a}')$ at the next time step was known for all possible actions $\mathbf{a}'$, then the optimal strategy is to select the action $\mathbf{a}'$ maximizing the expected value of $r + \gamma Q^*(\mathbf{s}', \mathbf{a}')$:

$$Q^*(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}'}\left[r + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}', \mathbf{a}') \,\middle|\, \mathbf{s}, \mathbf{a}\right] \tag{2}$$

The following clause proposes some examples of the inputs, outputs, and the rewards of the DRL network.

- **Inputs** (implying **network states**): The main factor of the network environment state is the user activity/demand distribution. It is assumed that the user location can be at least coarsely estimated either using the GPS information obtained from the location service or estimated with the base station ID and received arrival signal strength. The latitude-longitude coordinate is further converted to the pixel coordinate. Then, at the t-th time snapshot, a matrix is generated, expressing the user activity/demand map where the intensity at each matrix position (because the spatial-dependent measurements are collected in the form of matrix, hereafter matrix positions are used for pixel coordinates) is the measure of the user activities, e.g. data rate demand or quality of service class identifier (QCI). Let the matrix be denoted by $\mathbf{X}_t \in \mathbb{R}^{N_u \times N_v}$, where N_u and N_v denote the number of matrix positions of the two-axis converted from the latitude and longitude coordinate, respectively. Moreover, besides the user activity map, the matrix can also be extended to a multi-channel matrix to capture more network environment information, such as signal strength radio map. For example, a matrix $\mathbf{X}_t \in \mathbb{R}^{N_u \times N_v \times 2}$ can be formed, where the first $N_u \times N_v$ submatrix along the third dimension $\mathbf{X}_t(:,:,1)$ (hereafter the MATLAB matrix notation is used for the simplicity of denoting the entries of a matrix) implies the user activity map, while the second submatrix along the third dimension $\mathbf{X}_t(:,:,2)$ refers to the radio signal map. Each training input sample can be defined as matrix (or a multi-channel matrix) $\mathbf{X}_t$. By passing them through the convolutional layer the **spatial correlation** can be captured. The **temporal correlation** can also be captured by defining the input training sample as a sequence of the matrices $\{\mathbf{X}_\tau : \tau = t - T + 1, \ldots, t\}$ over T successive time snapshots $\tau = t - T + 1, \ldots, t$. To summarize, let the m-th sample be denoted by $\mathbf{X}^{(m)}$, then the two alternatives of input training samples are:

    - $\mathbf{X}^{(m)} \coloneqq \mathbf{X}_{t_m}$ to capture the **spatial correlation only** for the t_m-th time snapshot, where $\mathbf{X}_{t_m} \in \mathbb{R}^{N_u \times N_v \times K}$ and K is the number of considered network environment variables (e.g. user activity, or radio signal strength) that affect the system performance. Each input training sample is a $N_u \times N_v \times K$ image.

    - $\mathbf{X}^{(m)} \coloneqq \{\mathbf{X}_\tau : \tau = t_m - T + 1, \ldots, t_m\}$ to capture both the **spatial and temporal correlation** over T successive time snapshots. In this case, each input sample is a $N_u \times N_v \times (K \cdot T)$ matrix implying the values of selected network environment variables across spatial and time domain.

- **Outputs** (implying the predicted Q values corresponding to **actions**): The output layer is a fully connected linear layer with an output vector of the Q value (see Eqn. (3)) for all valid network optimization actions. A discrete network action space over the selected network control parameters is defined. As an example, one can consider (but not limited to) the following control parameters: transmit power level $p \in \mathcal{P} \coloneqq \{low, medium, high\}$, multiple input multiple output (MIMO) mode $d \in \mathcal{D} \coloneqq \{2 \times 2,\ 4 \times 4,\}$, frequency bands $b \in \mathcal{B} \coloneqq \{800MHz, 2.4GHz, 5GHz, 60GHz\}$. Then, the action space is $\mathcal{A} \coloneqq \mathcal{P} \times \mathcal{D} \times \mathcal{B}$ and there are $|\mathcal{P}| \cdot |\mathcal{D}| \cdot |\mathcal{B}| = 3 \cdot 2 \cdot 4 = 24$ possible actions, and the output is a vector $\mathbf{y} \in \mathbb{R}^{24}$. This output is a vector of the Q values corresponding to all 24 actions in the action space. Another option is that, instead of forming empirical searching space of the control parameters, one can also define some of the action subspaces based on the tuning directions, e.g. $\Delta_p = \{+\delta_p\, mW, -\delta_p\, mW\}, \Delta_d = \{+2 \times 2\ antennae, -2 \times 2\ antennae\}$, and the cardinality of the action space $\mathcal{A} \coloneqq \Delta_p \times \Delta_d \times \mathcal{B}$ is reduced to $|\Delta_p| \cdot |\Delta_d| \cdot |\mathcal{B}| = 2 \cdot 2 \cdot 4 = 16$. If there are multiple variables with many possible configuration values, defining some of the action subspaces based on the tuning directions could decrease the output state space. Moreover, one can also include the actions from the neighbouring network elements to incorporate the interaction between multiple agents, even though it may lead to a large output state space.

- **Rewards:** The resulting performance metrics (PMs) corresponding to the actions are considered in the rewards. The following mapping functions for defining rewards are proposed:

    - A continuous value for a target PMs, e.g. the service success rate.

    - A quantified measure for multiple Key Performance Indicators (KPIs). Since KPIs may have different scales, the KPIs can be normalizes to the same scale and obtain a weighted sum.

- Simplified feedback. One can clip all positive rewards at 1 and all negative rewards at -1, leaving the unchanged performance as 0. Or, to differentiate the magnitude of the performance, one can add more quantified levels of the positive and negative performance.

The DRL network provides an approximation of $Q^*(\mathbf{s},\mathbf{a})$, i.e. $Q(\mathbf{s},\mathbf{a};\boldsymbol{\theta}) \approx Q^*(\mathbf{s},\mathbf{a})$ where $\boldsymbol{\theta}$ is the parameter set characterizing the DRL network (e.g. weight matrices and bias vectors). One can train the network by adjusting the parameter $\boldsymbol{\theta}_\mathrm{i}$ at each iteration i to reduce the mean-squared error in the Bellman equation in (2), where the optimal target values $r + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}',\mathbf{a}')$ are substituted with approximated target values $y = r + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}',\mathbf{a}';\ \boldsymbol{\theta}_\mathrm{i}^-)$, where $\boldsymbol{\theta}_\mathrm{i}^-$ denotes the parameters from some previous iteration, for example, one can set $\boldsymbol{\theta}_\mathrm{i}^- = \boldsymbol{\theta}_{\mathrm{i}-1}$. This leads to a sequence of loss functions $L_i(\boldsymbol{\theta}_i)$ that changes at each iteration i, written as:

$$L_i(\boldsymbol{\theta}_i) = \mathbb{E}_{\mathbf{s},\mathbf{a},r,\mathbf{s}'}\left[\left(y - Q(\mathbf{s},\mathbf{a};\ \boldsymbol{\theta}_i)\right)^2\right]. \tag{3}$$

With the above loss function, the parameters $\boldsymbol{\theta}_i$ can be updated at each iteration i using the *gradient descent algorithm*. For more details of the deep Q-learning algorithm for training the parameters $\boldsymbol{\theta}$ readers can refer to Algorithm 1 in [i.60].

4.8.3.2.2 Transfer Learning Approach

What has been described so far tells us how to train a DRL model for an existing system. The next challenge is to define what and how to transfer the knowledge to a new system. In the following, the definition of transfer learning is given first.

Definition 1 (Transfer Learning [i.61]). Let a domain $D := \{X, P(X)\}$ consist of two components: a feature space X and marginal probability distribution $P(X)$. Let a task $T := \{Y, f(\cdot)\}$ consist of two components: a label space Y and an objective predictive function $f(\cdot)$. Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$.

In more details, one wants to answer the following questions: In case of $D_S \neq D_T$ or $T_S \neq T_T$, who receives the knowledge? How much knowledge to transfer? What information ought to be transferred? And how to adapt the transferred model in the new system?

To answer the former two questions, one needs to be aware that the transferability of learning strongly depends on the similarity between the source and target domains. In general, higher similarity or correlation between the source and target domains implies a better transferability [i.56], [i.59]. To answer the latter two questions, one needs to understand what the DRL model learns at different layers. At low/medium layers, it learns the designed reward (or loss) in response to the actions depending on the local edges and local shapes of the input maps (e.g. user demand map or/and radio signal map). In contrast, at high layers, it learns the dependence of the reward (or loss) on the higher-level statistical data features, e.g. the demand distribution. Therefore, it is meaningful to retain the low/medium layers of the model and to fine-tune or modify the high layers - when this is applied to a new system.

a) Parameters to Transfer

Assume that a pretrained DRL model is obtained in the existing system, and the parameters and the hyperparameters of the model are saved in a data file. To give an example of the parameters and the hyperparameters, the parameters are referred to the weight matrices and bias vectors between each two successive layers, and the hyperparameters include learning rate, number of hidden layers and units, number of filters and filter size in each convolutional layer, stride size of each max (or average) pooling layer, and the activation function. Suppose L layers are defined, a collection of parameters and hyperparameters for the l-th layer are denoted as $\Theta^{(l)}, l = 1, \ldots, L$ (e.g. including parameters such as weight matrix between l-th and $(l-1)$-th layer $\mathbf{W}^{(l)}$, bias vector $\mathbf{b}^{(l)}$, and hyperparameters such as type of the layer $\mathrm{T}^{(l)}$, number of the units of the layer $N_u^{(l)}$, filter size $S_f^{(l)}$, stride size $S_S^{(l)}$). Let the set of the general hyperparameters of the model be denoted by $\mathcal{H}$ (e.g. including learning rate α, loss function F, batch size S_b, activation function A). The proposition is to transfer a subset of the parameters and hyperparameters that characterizes the low/medium layers of the pretrained DRL model $\{\Theta^{(1)}, \ldots, \Theta^{(K)}\}$ where $K \leq L$ and the general hyperparameter set $\mathcal{H}$.

b) Transfer Criteria

Our motivation is as follows. The more similar two models (either the mathematical model of the optimization functions or physical model of the communication network systems) are, the more knowledge extracted from one system can be utilized to derive a new model of another system. Therefore, the similarity measure can be used to decide:

- the number of low/medium layers to be transferred from a pretrained optimization function to another function in Scenario I as shown in Figure 29; or
- the number of low/medium layers to be transferred from an existing network system to a new system in Scenario II as shown in Figure 30.

Figure 29 and Figure 30 describe it in more in details.

In Scenario I (shown in Figure 29), given a pretrained optimization model, for a pretrained self-organization network (SON) function for coverage and capacity optimization, one can reuse the lower/medium layers of the pretrained model, and modify the higher layers based on a difference set of control parameters and objective rewards or costs of another SON function, e.g. mobility robustness optimization. By training the revised layers of the DRL network only with newly collected samples, one can derive the new optimization function of a different SON function.

In Scenario II (shown in Figure 30), the goal is to learn an optimization function of a newly deployed system from an existing system, for example, by transferring the knowledge extracted from an existing BS to a newly deployed BS. To this end, one can keep the lower/medium layers of the pretrained model and retrain the high layers by feeding the DRL network with samples collected from the new BS.

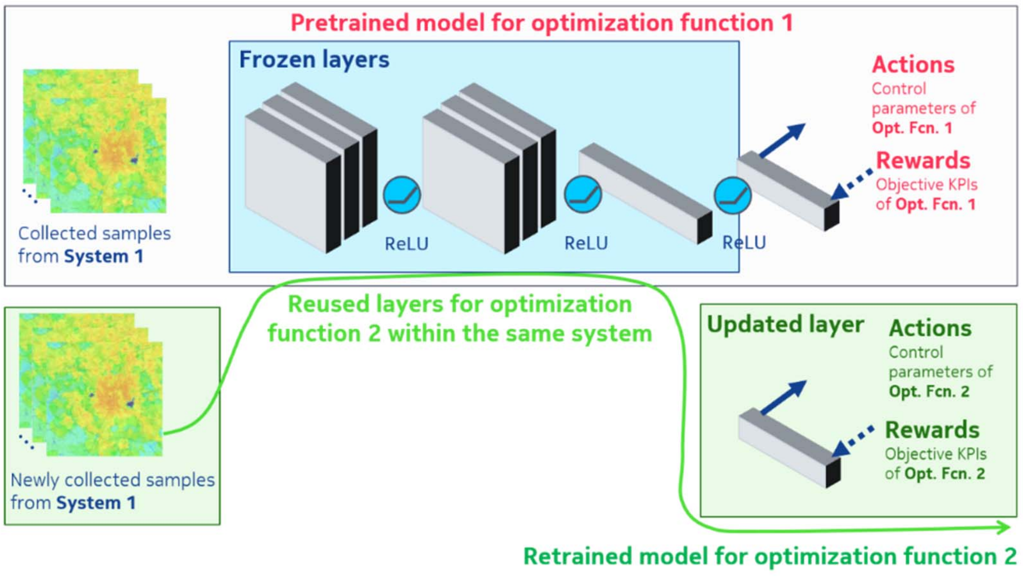
Pretrained model for optimization function 1
Frozen layers
Collected samples from System 1
ReLU
ReLU
ReLU
Actions
Control parameters of Opt. Fcn. 1
Rewards
Objective KPIs of Opt. Fcn. 1
Reused layers for optimization function 2 within the same system
Newly collected samples from System 1
Updated layer
Actions
Control parameters of Opt. Fcn. 2
Rewards
Objective KPIs of Opt. Fcn. 2
Retrained model for optimization function 2

Figure 29: Example of Scenario I, transferring knowledge across network optimization functions

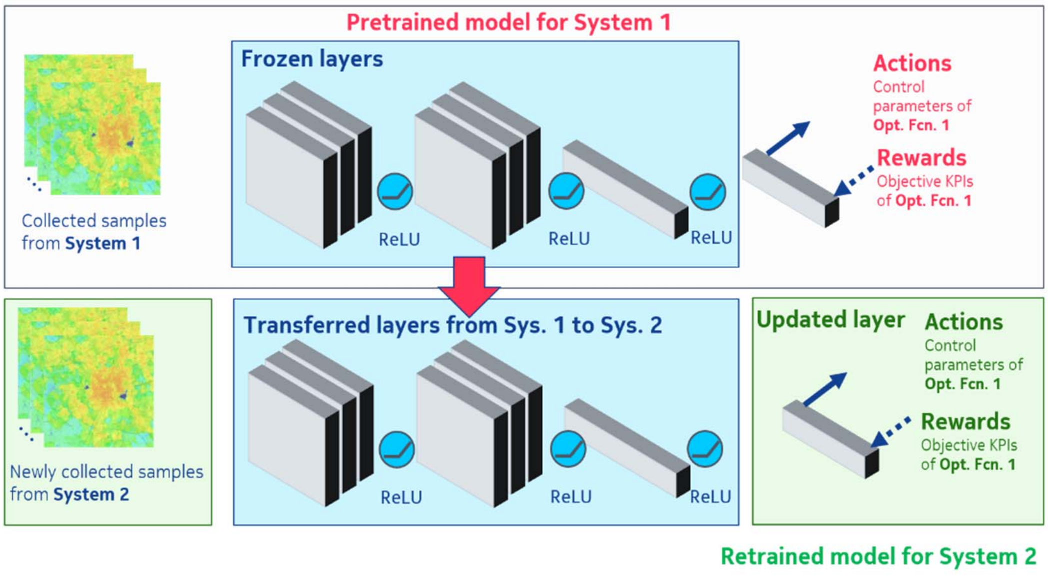
Pretrained model for System 1
Frozen layers
Collected samples from System 1
ReLU
ReLU
ReLU
Actions
Control parameters of Opt. Fcn. 1
Rewards
Objective KPIs of Opt. Fcn. 1
Newly collected samples from System 2
Transferred layers from Sys. 1 to Sys. 2
ReLU
ReLU
ReLU
Updated layer
Actions
Control parameters of Opt. Fcn. 1
Rewards
Objective KPIs of Opt. Fcn. 1
Retrained model for System 2

Figure 30: Example of Scenario II, transferring knowledge across network systems

4.8.4 Implications

Conventional machine learning algorithms have been designed to work in isolation and to be trained for specific tasks. Once the domain (defined by the feature space and the probability distribution over this space) changes, or the learning task is different, the models need to be rebuilt from scratch. Our proposed transfer-learning based network automation solution provides a means to overcome the isolated learning paradigm and help utilize knowledge acquired for one task to solve the related ones. Here, the knowledge could be, for example, the pre-trained models of network functions for self-learning, self-healing, and self-optimization. Note that, while many conventional network automation solutions improve the resource utilization efficiency by sharing physical resource (e.g. time slots, frequency bands) using various multiplexing techniques, in this case "resource efficiency" refers to the effort spent on obtaining knowledge, as well as the storage space of the obtained knowledge. Thus, the goal is to create new solutions for network automation, where efficiency of knowledge usage can be improved by "knowledge multiplexing", such that pre-trained knowledge can be reused and shared among multiple network elements.

To this end, a network knowledge sharing framework needs to be designed, enabling the network elements to interact with each other, and guaranteeing unhindered flow of information among them about the knowledge learned from the network. Below is a list of some features essential to the network knowledge sharing platform:

- Knowledge availability: A network element can be quickly notified whether other network elements hold useful knowledge for it, and, if so, which network element to communicate to access the information.
- Accessibility and distributivity: Analogue to the content sharing platform, the storage of knowledge should be distributed among a subset of the network elements in a way that an element can access the knowledge as quickly as possible, while keeping limited copy of the knowledge to reduce the storage cost.
- Interactivity: Knowledge sharing is no longer a one-way process. Communication between network elements allows the network elements as knowledge receiver to request "personalized" knowledge. It should also allow feedback to the knowledge providers to enlarge/improve their knowledge.
- Smart model indexing: To efficiently store, track, and locate the models, proper labelling of the models and classifying them into various groups are required.
- Dynamic analytics: This feature allows the operators to keep a track of the dynamic topology of the sharing network: which model is the most reusable? Which network elements learn and contribute the most? Is there new knowledge appearing and being learned? Which network elements spend resource on learning redundant information? Such analytics can be further used to optimize the knowledge sharing framework.

Apart from designing the knowledge sharing framework, more fundamental study needs to be done to solve the two major challenges in transfer learning:

- Negative transfer: It refers to scenarios where the knowledge transfer from the source to the target leads to a drop in the network performance of the target task. The reason could be that the source domain/task are not sufficiently related to the target domain/task, or that the transfer learning method could not capture the relationship between source and target domain/task well. Robust transfer mechanisms need to be designed to avoid negative transfer.

- Transfer bounds: It is also essential to quantify the transferability in the sense of quality of the transferred model and resource costs for knowledge transfer.

4.8.5 Proof of concept

The benefits of transfer learning techniques have been assessed on a couple of different use cases and published in two conference papers. In [i.62], it is investigated the possibility of performing a knowledge transfer for the prediction of tilt-dependent received signal strength maps. Two cases are considered:

1) where the knowledge acquired for a particular antenna tilt setting is transferred to a different tilt configuration of the same antenna; or

2) a different antenna with the same tilt configuration. Promising results supporting knowledge transfer are obtained through extensive experiments conducted using different machine learning models on a real dataset.

In [i.63], transfer learning is exploited for channel quality prediction across different wireless channels. Namely, it is proposed a framework based on transfer learning to predict the channel quality of a given frequency carrier when no or minimal information is available on the very same frequency carrier for model training. For the transfer learning task, convolutional neural networks and long short-term memory networks are used. The performance evaluation carried out on the reference dataset demonstrates the validity of the proposed transfer learning approach.

4.8.6 Relevance for ZSM

As shown in the above clauses, Transfer Learning represents a key enabler for network and service automation, in particular to increase the efficiency and in support to the planning, deployment and optimization tasks. Transfer learning is therefore relevant and of importance to ZSM.

ZSM should investigate further how to enable the integration of Transfer Learning and its mechanisms within the ZSM architectural framework, allowing a flexible and versatile use of the technique. This could further take the form of applicability studies and use cases and identifying which elements of the approach could and should be standardized. It could also take the form of investigation to understand and document how Transfer Learning can become an efficient tool assisting the human operator in various operational scenarios (e.g. decision support for new infrastructure deployment or dynamic optimization, forecasting of gain and critical parameters to consider when/before applying Transfer Learning in given situations, etc.)

Annex A:
Change History

Date	Vers ion	Information about changes
28.February 2018	0.0.1	Skeleton
6. March 2018	0.0.2	Add first none approved contributions
15. March 2018	0.0.3	Extract none approved contributions, simplify Skeleton
1. October 2018	0.0.4	Split Skeleton for Atomic and Composite Means of Automation according to ZSM(18)000239r4_ZSM-005_Split_of_Elements_and_Solutions_for_Automation.docx
2. November 2018	0.1.0	Incorporated approved contributions after meeting ZSM#4: ZSM(18)000196r2_ZSM005_MoA_for_Intent_Based_Orchestration.docx ZSM(18)000184r4_ZSM005__Policy_Driven_Automation.docx
10. December 2018	0.1.1	Incorporated approved contribution after meeting ZSM#5: ZSM(18)000558_ZSM005_Intent_Based_Means.docx
2. May 2019	0.1.2	Incorporated approved contributions after ZSM-Interim#03 and ZSM-5 tech calls: ZSM(18)000192r2_Introduction_to_Means_of_Automation.docx ZSM(18)000518r2_ZSM005_Means_of_Automation_for_Network_Stability.docx ZSM(18)000517r2_ZSM005_Means_of_Automation_for_Network_Governance.docx
8. September 2019	0.1.3	Incorporated approved contribution after ZSM-7c tech call: ZSM(19)000269r3_ ZSM005_MoA_Transfer_Learning.docx Changes towards stable draft: Consolidated References in clause 2.2 Added various Editor's Notes Deleted clause 3 (Definitions, Symbols etc) and renumbered later clauses

1. December 2019	0.1.4	Incorporated approved contributions:
		ZSM(19)000468r1_ZSM005__MoA_Reinforcement_Learning.docx (appr. ZSM-08b)
		ZSM(19)000473r2_ZSM005_Contributions_Overview.docx (appr. ZSM-08)
		Consolidated References in clause 2.2

11. December 2019	0.2.0	Incorporated approved contribution and changes: ZSM(19)000639r2_ZSM005_Contributions_Overview_Augmented.docx (appr. ZSM#09) Updates for stable draft: Removed empty editorial clause Revised references Resolved editorial Editor's notes
6. January 2020	0.3.0	Overall Updates after complete review, final updates in ZSM#9a. Specific updates for clauses 4.4.4 - 4.4.6 and 4.5.5 - 4.5.6. Complete removal of Editor's note.
19. February 2020	0.4.0	Substitute of all "must" and "shall".
1. March 2020	0.4.1	Improve substitution

History

Document history		
V1.1.1	May 2020	Publication