Orchestration Procedures for the Network Intelligence Stratum in 6G Networks

Livia Elena Chatzieleftheriou, Marco Gramaglia, Miguel Camelo, Andres Garcia-Saavedra, Evangelos Kosmatos, Michele Gucciardo, Paola Soto, George Iosifidis, Lidia Fuentes, Gines Garcia-Aviles, Andra Lutu, Gabriele Baldoni, Marco Fiore

***Abstract*—The quest for autonomous mobile networks introduces the need for fully native support for Network Intelligence (NI) algorithms, typically based on Artificial Intelligence tools like Machine Learning, which shall be gathered into a NI stratum. The NI stratum is responsible for the full automation of the NI operation in the network, including the management of the life-cycle of NI algorithms, in a way that is synergic with traditional network management and orchestration framework. In this regard, the NI stratum must accommodate the unique requirements of NI algorithms, which differ from the ones of, e.g., virtual network functions, and thus plays a critical role in the native integration of NI into current network architectures. In this paper, we leverage the recently proposed concept of Network Intelligence Orchestrator (NIO) to (i) define the specific requirements of NI algorithms, and (ii) discuss the procedures that shall be supported by an NIO sitting in the NI stratum to effectively manage NI algorithms. We then (iii) introduce a reference implementation of the NIO defined above using cloud-native open-source tools.**

***Index Terms*—Network Intelligence, Network Intelligence Orchestration; Intelligence Plane**

I. Introduction

One of the key expectations for 6G networks is definitively closing the gap to full autonomous operation, by enabling self-configuration with minimal or no human intervention through the adoption of intelligent algorithms. This calls for the introduction of an additional *stratum* in the network, i.e., one layer that is specifically devoted to the management and orchestration of the intelligence in the network [1].

Managing intelligence in the network. Artificial Intelligence (AI), and more concretely Machine Learning (ML) techniques, pushed by the increased availability of measurement data and computational resources within mobile networks, have the potential to develop the Network Intelligence (NI) that will enable fully autonomous 6G networks. Practical steps towards this direction have been recently investigated by major Standard-Defining Organization (SDO) entities such as 3GPP and the European Telecommunications Standards Institute (ETSI), as well as by global industrial initiatives like O-RAN. Within the EU-funded DAEMON project [2],

L.E. Chatzieleftheriou is with IMDEA Networks Institute and University Carlos III de Madrid (UC3M). M. Gramaglia is with UC3M. M. Camelo-Botero and P. Soto are with University of Antwerp - imec A. Garcia-Saavedra is with NEC Laboratories Europe. E. Kosmatos is with WINGS ICT Solutions. M. Gucciardo and M. Fiore are with IMDEA Networks Institute. G. Iosifidis is with Technical University of Delft. L. Fuentes is with University of Malaga. G. Garcia-Aviles is with i2CAT Foundation. A. Lutu is with Telefonica Research. G. Baldoni is with Zettascale Technology.

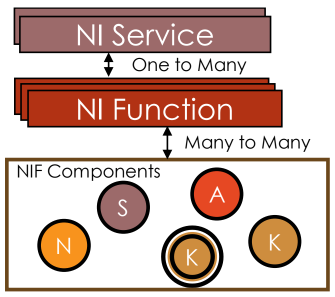


Fig. 1. The hierarchical taxonomy of NI algorithm components. A NIF corresponds to an individual NI instance that assists a specific functionality: for example, it could capture the implementation of a capacity forecasting task, assisting an NI edge orchestration functionality.

we have identified key gaps in the above efforts. Namely, the implementations currently provided by the standarization entities do not deliver the following essential functionalities: (i) coordination of the NI across different network domains; (ii) data management across NI instances in a decentralized and unified fashion; (iii) support for the NI lifecycle management; and, (iv) solid methodologies for the definition and representation of NI models. Failing to meet the above would severely compromise the applicability of NI and, subsequently, its practical adoption within 6G networks.

Network Intelligence stratum. As part of its activities, the DAEMON project aims at filling the gaps above by presenting a clear set of functional and non-functional requirements that target the coordination of NI instances in an end-to-end fashion. To this end, we are proposing a new *Network Intelligence stratum* that complements and interacts with the existing planes in current and next-generation mobile networks, i.e, the user/data, control, and management planes. In our effort to define the NI stratum organization and operations, we already introduced a reference representation of complex NI algorithms as a hierarchy, sketched in Fig. 1, of Network Intelligence Services (NISs) that can be broken down into one or more Network Intelligence Functions (NIFs), which is turn are composed of atomic NIF Components (NIF-Cs) [3]. We also specified how NISs and NIFs can be managed by a Network Intelligence Orchestration (NIO) with a precise internal structure of fundamental building blocks [3].

In addition, in a separated work, we defined a suitable reference representation to be adopted by the NIO to model any NI algorithm [4]. To that end, we adapted a popular model that is widely adopted for autonomous and self-adaptive

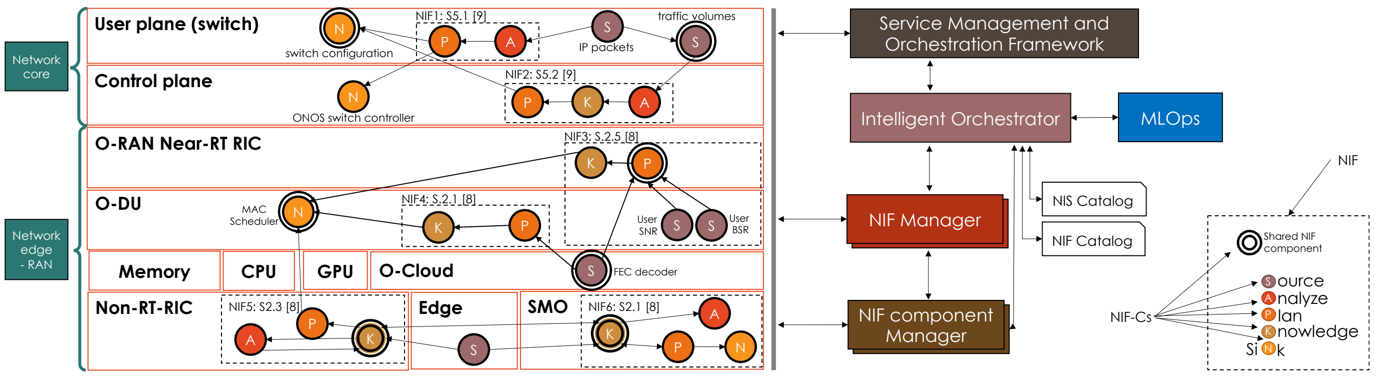


Fig. 2. NI-native architectural concept proposed by the DAEMON project for the NI stratum. The diagram portrays the interactions between many different NIFs that implement two NI-assisted functionalities, or NIS, also developed in the project. The NIF-Cs that compose each NIF are categorized using our original N-MAPE-K representation. The hierarchies of NISs, NIFs, and NIF-Cs are managed all at once by the NIO framework, by avoiding conflicts and leveraging synergies among them.

systems, i.e., the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) feedback loop [5]. Building on top of the MAPE-K representation, we dissected NI algorithms into common elements that have different characteristics (e.g., a data-gathering probe or a Neural Network model), and introduced original training and closed control loops that a NIF may implement, which resulted into an extended Network MAPE-K (N-MAPE-K) model tailored to the NI environment. The N-MAPE-K model allows capturing (i) the inference loop, (ii) a traditional supervised training loop, and (iii) a second training loop dedicated to online learning.

Mapping NI algorithm components into the N-MAPE-K representation allows highlighting the following fundamental classes of atomic NIF-Cs.

- *Sensor* NIF-Cs specify all the probes needed to gather the input measurement data.
- *Monitors* NIF-Cs specify how each NIF interacts with the Sensor NIF-Cs and gathers raw data from them.
- *Analyze* NIF-Cs include any pre-processing, summary, or preparation of the data for the specific NI algorithm implemented in the plan NIF-Cs.
- *Plan* NIF-Cs constitute the specific NI algorithm implemented by the NIF.
- *Execute* NIF-Cs specify how the algorithm is going to interact with the managed system and how to possibly change its configuration parameters.
- *Effector* NIF-Cs specify the configuration parameters updated in the Network Function (NF), and the Application Programming Interfaces (APIs) to be used to that end.

Contribution. In this paper, we present a unified framework that brings together our earlier proposals for (i) the operational hierarchy of NI components in the NIO, and (ii) the N-MAPE-K representation of NIF-Cs. By doing so, we make a step forward the vision of a complete NI stratum anticipated above.

An illustrative example of the resulting integration is provided in Fig. 2. There, each circle depicts a NIF-C and a double circle captures a NIF-C shared among multiuple NI-assisted functionalities. For example, the circle in the O-Cloud rectangle captures the FEC decoder. Multiple united NIF-Cs constitute a NIF, e.g. Nuberu [6] or Henna [7], to mention two solutions developed in the project itself. Finally, by combining NIFs we get a NIS: as an example, the integration of different RAN-related algorithms can realize an overall reliable virtualized RAN (vRAN) service.

We identify and present in detail the specific requirements that NI algorithms pose on the NIO framework discussed before, understanding their specificity and devising procedures that the NIO shall provide in §II. Then, in §III, we analyse how the NIO can support them by discussing specific NIP processes. Finally, in §IV, we discuss how state-of-the-art open-source solutions for cloud-native orchestrations can be leveraged to implement our proposed framework.

II. The need for specific NI stratum procedures

The concurrent instantiation of many different NIFs raises challenges that the architecture we propose allows addressing. Next, we detail the management needs that such challenges create, and exemplify them with representative NI-assisted functionalities developed in the DAEMON project [8], [9].

A. *Conflict resolution*

DAEMON's NIO allows to efficiently re-use and combine different elements that can be shared across NIFs, by representing their split into atomic NIF-Cs that abide by the N-MAPE-K framework [4]. This eventually enables building in an effective ways a NIS, analogously to the approach used by 3GPP SA5 to build the Network Slicing data model –where a Network Slice is decomposed into Network Slice subnets. However, while composing NIFs to build a NIS, through the sharing of different NIF-Cs, possible conflicts on operations and/or resources may arise. It is hence a task of the NIO to arbitrate the operation of such components, guaranteeing that the overall goal of the NIS is met.

Let us illustrate this issue by detailing the arrangement of Nuberu and Athena, two NIFs described in [8] (§2.1 and §2.5, respectively), that aim at improving the resiliency of a virtualized radio access network (vRAN) system by acting on MAC scheduling decision at the Distributed Unit (DU) of base stations. Nuberu [6] proposed a re-design of the full stack to be cloud-native and resilient; while Athena introduced a model that learns the limits of the infrastructure and takes scheduling decisions. Thus both algorithms support the Radio MAC scheduler acquiring knowledge from similar input data (e.g., the information about the channel) and enforcing radio scheduling decisions, optimizing the reliability of the system, at different time scales. This results in the sharing of two NIF-C, the sources and the sinks between these two NIFs, as also shown by the N-MAPE-K representation depicted in Fig. 2.

A similar consideration applies when dealing with mixed user and control intelligence, as in the case of the algorithms in §5.1 of [8], whose goal is (i) performing in-switch inference at line rate [7] and (ii) achieving optimal configuration of circuit switching by using real-time traffic demands. The NIF implementing in-switch inference, i.e., NIF1 in Fig. 2, acts almost entirely in the user plane, directly classifying IP traffic and directly enforcing decisions into the NF that is classifying the traffic, the switch controller in this case. NIF2 in the figure generates instead the circuit switching configuration in the control plane and enacts it in the user plane. The configuration decision is taken based on information about traffic volume, which is available at each switch.

The previous two examples, in which different NIFs share sources and sinks, motivate the need for monitoring and coordination of policy enforcement. In this context, different conflicts may arise, as follows.

- *Conflicts when monitoring data.* Algorithms may need data from the same source but with different granularity. Hence, the NIF Manager shall guarantee that the required information arrives from the Sources to the specific Plan/Analyze modules with the necessary granularity (e.g., at subframe or packet level) in an automated manner to, e.g. avoid duplicating the monitoring over IP packets.
- *Conflicts in the policy enforcement.* Different NI algorithms may act on the same network functions (in the proposed example, the DU MAC scheduler), configuring different parameters. Thus, the Intelligent Orchestrator shall deploy conflict resolution policies with the NIF-C of each NIF to guarantee that, e.g., the scheduled MAC frame never exceeds the available capacity or contrasting selected users.

Therefore, **the NIO shall oversee and amend any suboptimal decision taken by individual NIFs** by closely monitoring the access to data sources and the policies determined by decision-making algorithms.

B. Knowledge sharing among NIFs

Fig. 2 also illustrates the shared representation of two NIFs detailed in [9] (§2.3 and §2.1, respectively): energy-driven vRAN orchestration, i.e., NIF5 in the figure, and energy-aware VNF placement, or NIF6 in the figure. In the case of these two NIFs, energy consumption measurements from an edge cloud platform are required and a source node component is hence shared. Moreover, NIF5 generates knowledge about high-performing RAN control policies given a context and once virtualized instances of RAN components have been deployed. On the other hand, NIF6 is in charge of VNFs placement, which in this case implements virtualized RAN functions. In this context, **the NIO shall provide centralized coordination among multiple NIFs**. Such centralized coordination would allow sharing of knowledge that fostered synergetic performance improvements between both NIFs. For instance, part of the knowledge learned by NIF6 can be used by NIF5 to make better placement decisions and, vice versa, some knowledge learned by NIF5 can be used by NIF6 to enforce informed (placement-aware) RAN control policies.

Knowledge sharing aspects should also be available cross-domain. For instance, in §4.2 of [9] we describe an anomaly detection solution for IoT platforms. In that scenario, the user plane traverses multiple domains, which brings new challenges in terms of running root-cause analysis of anomalies. Hence, the parties involved in building the user plane for the IoT devices suffering from anomalies should be integrated into the anomaly detection scheme, and such synchronization shall happen at the NI Orchestration.

C. Model selection, catalog, and re-training

Although this is not a condition directly stemming from the design of the NI algorithms themselves, NISs may need to build on the knowledge of the underlying environment. This calls for awareness of the software/hardware environment (e.g., as the performance of a specific FEC implementation depend on the target hardware [10]) or of the location of the device where they are executed (e.g., as reconfigurable intelligent surfaces may have different behaviors according to their geographical position and surrounding environment [11]).

When executed in the context of a pure ML environment, these tasks are natively tackled by several MLOps frameworks. In the context of an NI-native architecture, however, this requires tight interaction with the underlying orchestration environment. To guarantee that the deployed NIF can operate in the right context, NI models need to match the specific hardware-software-environmental characteristics of the network functions deployed in a network service. Thus, **the NIO shall exchange execution context information with the sibling MANO operating in the network**, so as to select the proper model to be used for inference within a NIF.

This incidentally calls for the need of a model catalog from which the NIO can select the most appropriate model depending on the specific infrastructural status operated by the network at a certain point in time. If no model is available for the specific execution environment, the NIO shall be able to invoke the training of a new model, fetching the required data as required by the target algorithm.

III. NI-NATIVE ARCHITECTURAL PROCEDURES

As described in the previous section, several considerations and challenges emerge while concurrently deploying multiple NIFs providing the same or different NISs. Building on the NIO organization and N-MAPE-K represenation of NIF-Cs, we next define processes that answer such needs.

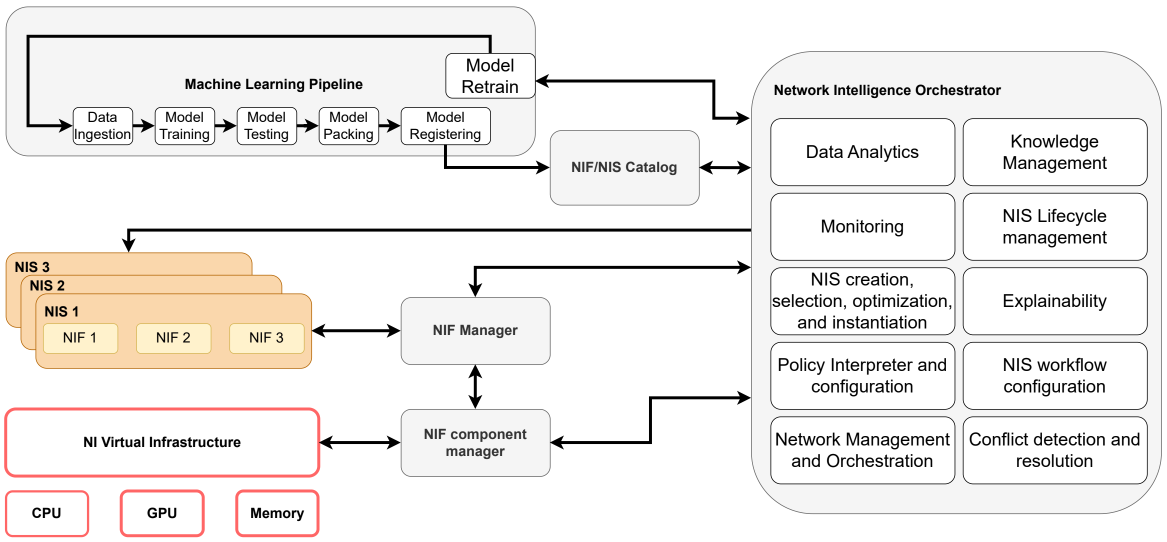

Machine Learning Pipeline
Model Retrain
Data Ingestion
Model Training
Model Testing
Model Packing
Model Registering
NIF/NIS Catalog
Network Intelligence Orchestrator
Data Analytics
Knowledge Management
Monitoring
NIS Lifecycle management
NIS creation, selection, optimization, and instantiation
Explainability
Policy Interpreter and configuration
NIS workflow configuration
Network Management and Orchestration
Conflict detection and resolution
NIS 3
NIS 2
NIS 1
NIF 1
NIF 2
NIF 3
NIF Manager
NI Virtual Infrastructure
NIF component manager
CPU
GPU
Memory

Fig. 3. The NI stratum and the functional blocks of the Network Intelligence Orchestrator.

A. Rationale

When used outside the network domain, the set of solutions that deal with the lifecycle management of intelligent algorithms is usually referred to as MLOps [12]. Items such as Feature Engineering, Model Training, Model Engineering, as well as their integration in a CI/CD system are usually encompassed in this definition. When transferring this view into the mobile network realm, however, these items cannot be transferred *as is*, mostly because of the very different timescales that are usually involved in network environments, which may go down to sub-ms levels.

Therefore, we split the items into elements that are only related to pure ML tasks and are commonly executed offline, either only once or very rarely. We mark them as *Machine Learning Pipeline* in Fig. 3. Instead, other elements need to directly interact with the NIFs in the network, continuously evaluating the quality of the NIS and performing fine-grained lifecycle management of the NIF-Cs, including their coordination. These are the most interesting in the context of the NI stratum, and we discuss them next.

B. Overall description

As mentioned in the previous section, the NIO should incorporate multiple functionalities to support the described challenges and beyond. Some key functionalities are shown in Fig. 3. Their main purposes are as follows.

- **Data analytics.** This block includes any pre-processing or preparation of the data (e.g., averages, autoencoders, filtering, or clustering algorithms).
- **Knowledge management.** A critical components of the NIO, the knowledge management block provides all the mechanisms required to plan, organize, act, and control the knowledge across all the deployed NIS.
- **Monitoring.** This block processes the NIS's information. As NIS can be composed of both non-ML (e.g., traditional VNFs) and ML-based functionalities, the monitoring information can also be of both types: ML-related (e.g., model-specific metrics and detection of data drift for essential features), and non-ML-related (e.g., QoE, QoS, etc.). In addition, this block will monitor NIs in both training and inference deployments.
- **NIS lifecycle management.** This functional block takes care of the deployment and maintenance of working ML models, aligned with MLOps practices. This includes the creation of new ML pipelines to re-train ML models.
- **NIS creation/selection, optimization, and instantiation.** Before any deployment, the NIO has to select (e.g., based on hardware constraints), optimize (e.g., compress a Neural Network (NN)-based NIS to achieve a given trade-off between model size and performance), and instantiate the selected NIS. In case a given NIS is not available in the catalog, the NIO should be able to create it based on the available data and execution context information.
- **Model explainability.** This block provides the methods that help human experts understand NIS composed of black-box (e.g., deep neural network) ML algorithms. This is a fundamental capability to understand the cause of a decision from a NIS such that a human can consistently review/correct its results.
- **Policy interpreter and configuration.** This functional block interprets high-level user's intent objectives, e.g., high-level QoE targets and business KPIs, that are associated with different NIS. If needed, it also performs changes in the policy.
- **NIS workflow configuration.** This block puts together data engineering, ML, and DevOps in a more straightforward, efficient, and effective fashion. In a general perspective, the NIO uses NIS workflow configuration to operationalize the deployment, monitoring, and lifecycle management in a modular and flexible way.
- **Network management and orchestration (MANO).** This functional block takes care of the lifecycle manage-

ment of the traditional Network Virtual Functions (NVF) that communicate with a NIS. In addition, it provides the context execution information from the network.
- **Conflict detection and resolution.** This block provides a mechanism to solve trade-offs that may emerge from conflicting objectives in the control and user planes, e.g., in establishing policies (at small timescales) versus enforcing such policies (at large timescales). This functionality allows the NIO to compare policies among different NIS to detect conflicts and perform conflict resolution based on comparison and resolution rules.

The previous functionalities provide the mechanisms that allow the NIO to address the challenges that can emerge when NISs are deployed across different network domains and operating in multiple time scales. Next, we briefly describe how the combination of some functional blocks can help to address the challenges described in the previous section.

C. Conflict resolution

We introduced two specific conflict cases in the previous section: (i) when conflicts emerge when monitoring data, e.g., algorithms may need data from the same source but with different granularity, and (ii) when conflicts in the policy enforcement of different NI algorithms may act on the same network functions but configuring different values for the target parameters. In situations like those above, the policy interpreter and configuration block will gather information about the policy guiding the different NIS and pass their interpretation to the conflict detection and resolution module. In both cases, a conflict will be detected, and the NIO will identify and apply the conflict resolution rules associated with (i) multi-time scale coordination and (ii) parameter constraints and execution priority. The outcome after applying the rules should provide a plan that will trigger configuration modification of the NIS policies. In the case of NIS empowered by black-box ML algorithms, the model explainability block will interpret policies associated with such algorithms.

D. Model selection, catalog, and re-training

NI solutions stored in the NIS/NIF catalog are inherently trained on hardware and software platforms that may not match the ones available in the new environment where they need to be deployed. In such cases, the NIS creation/selection, optimization, and instantiation block will obtain networking and execution context information from its MANO block operating in the network and select the proper model to be used in inference within a NIF. Suppose a mismatch between trained and targeted hardware/software appear. In that case, the same block should perform the optimization/adaptation (e.g., compression of a neural network, change of inference library from GPU to CPU) to match the new environment. In case no model is available for the specific execution environment, the NIS creation/selection, optimization, and instantiation block will create a new NIS and then notify the NIS workflow configuration block to trigger a new training phase.

E. Knowledge Sharing

NISs deployed in the same or across different domains use their knowledge to derive their execution plans. The knowledge management block will allow the NIO to understand the knowledge of each NISs, via the interaction with the model explainability block and derive new policies that represent the shared knowledge among NISs, by interacting with the policy interpreter and configuration block.

IV. Reference implementation

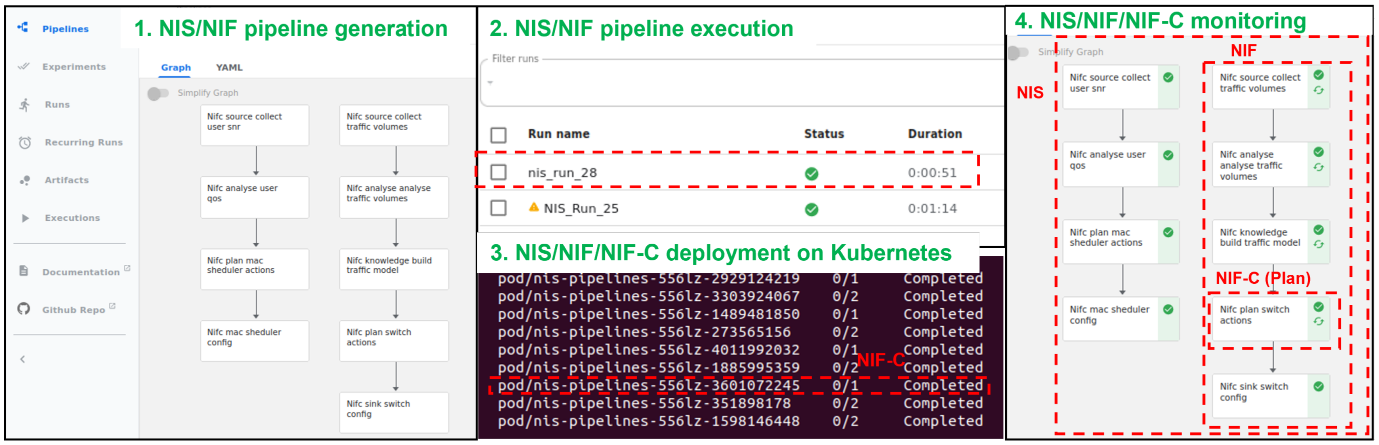
We implement the NI-native architecture presented above as a prototype using Kubernetes [13] as the main deployment environment. In addition, Kubeflow [14] is used to perform MLOps and as the baseline for developing some of the NIO functionalities. Furthermore, selected functionalities of the NIO are developed from scratch. The Eclipse Zenoh framework [15] is used for data flow programming among the NIF-Cs and for metric collection and aggregation, such as the ones coming from the sources NIF-Cs. A visual representation of the prototype implementation is in Fig. 4.

In the prototype, Kubernetes serves as the main deployment environment taking care of the MANO functionalities on top of a virtualized infrastructure. The Kubeflow deployment is realized as a Kubeflow cluster with one controller and 3 worker nodes, in which the NIF-C components are deployed as pods. The management of NIF-Cs is realized by the NIF component manager in Fig. 3 through the Kubernetes API. As described in previous sections, a set of interconnected NIF-Cs following the N-MAPE-K representation compose a NIF. This is realized by a pipeline of pods managed by the NIF Manager utilizing the Kubeflow Pipelines SDK. This is illustrated in Fig. 2. The generated pipeline of NIF-Cs is defined in Python, is translated in YAML, and then deployed in Kubernetes (both pods and connectivity) using the developed service which utilises the Kubeflow pipeline service. In the same fashion, NI Orchestrator manages the NISs (using Kubeflow) at a higher hierarchical level.

Following the described approach, we can provide a set of NI Orchestrator functionalities including (i) NIS composition, (ii) NIS lifecycle management, (iii) NIS workflow configuration, (iv) NIS selection, and (v) Monitoring, which are realized by building on the functionality already available in the Kubeflow framework. The developed monitoring service of NI Orchestration provides monitoring of (i) NIF-C/NIF/NIS deployment status, (ii) NIF/NIS pipeline progress, (iii) MLOps progress, (iv) resource utilization, and (v) performance KPIs. The rest of the functionalities are planned to be developed as separate modules integrated with the final solution.

The MLOps operations responsible for the model retraining, at the top of Fig. 3, is realized as ML pipelines in the Kubeflow environment, while the NIF/NIS catalog is created using a Docker repository linked to the Kubernetes environment.

Finally, it is important to stress that the NIF-C taxonomy (i.e., Analyse, Plan), as well as the adopted communication paradigm (Eclipse Zenoh) were adopted in all components of the architecture including NIF/NIS Catalogs (Dockers with


1. NIS/NIF pipeline generation
Pipelines
Experiments
Runs
Recurring Runs
Artifacts
Executions
Documentation
Github Repo
Graph
YAML
Simplify Graph
Nifc source collect user snr
Nifc source collect traffic volumes
Nifc analyse user qos
Nifc analyse analyse traffic volumes
Nifc plan mac sheduler actions
Nifc knowledge build traffic model
Nifc mac sheduler config
Nifc plan switch actions
Nifc sink switch config
2. NIS/NIF pipeline execution
Filter runs
3. NIS/NIF/NIF-C deployment on Kubernetes
NIF-C
4. NIS/NIF/NIF-C monitoring
Simplify Graph
NIS
NIF
NIF-C (Plan)
Nifc source collect user snr
Nifc source collect traffic volumes
Nifc analyse user qos
Nifc analyse analyse traffic volumes
Nifc plan mac sheduler actions
Nifc knowledge build traffic model
Nifc mac sheduler config
Nifc plan switch actions
Nifc sink switch config

Run name	Status	Duration
nis_run_28	✓	0:00:51
NIS_Run_25	✓	0:01:14

```
pod/nis-pipelines-556lz-2929124219   0/1    Completed
pod/nis-pipelines-556lz-3303924067   0/2    Completed
pod/nis-pipelines-556lz-1489481850   0/1    Completed
pod/nis-pipelines-556lz-273565156    0/2    Completed
pod/nis-pipelines-556lz-4011992032   0/1    Completed
pod/nis-pipelines-556lz-1885995359   0/2    Completed
pod/nis-pipelines-556lz-3601072245   0/1    Completed
pod/nis-pipelines-556lz-351898178    0/2    Completed
pod/nis-pipelines-556lz-1598146448   0/2    Completed
```

Fig. 4. Prototype demonstrating NIS/NIF/NIF-C pipeline generation, deployment and monitoring.

different prebuilt libraries per NIF-C type) and during the NIF/NIS creation process (different preconfigured attributes per NIF-C type). This taxonomy is further used by the solution for realizing tasks related to the application of policies and conflict resolution.

V. Conclusions

In this paper, we discuss the Network Intelligence (NI) orchestration procedures that are needed to support a NI-native architecture. Following the recent proposals, we identify the requirements that a range of NI algorithms based on AI techniques pose to the Network Intelligence Orchestration (NIO) framework. Based on this discussion, we devise requirements stemming from the concurrent instantiation of NI Services (NISs) and NI Functions (NIFs) that the NIO shall meet, as well as detailed processes that should be supported by the NIO framework. Finally, we go one step forward, and provide the discussion of a reference implementation of the NIO framework, building on top of state-of-the-art open-source tools for cloud-native orchestration.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 101017109 "DAEMON".

References

[1] 5G PPP Architecture Working Group, "The 6G Architecture Landscape - European perspective," Dec. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7313232
[2] "H2020 ICT-52 DAEMON," https://h2020daemon.eu/, 2021-2023, accessed: 2018-12-06.
[3] M. Camelo, M. Gramaglia, P. Soto, L. Fuentes, J. Ballesteros, A. Bazco-Nogueras, G. Garcia-Aviles, S. Latré, A. Garcia-Saavedra, and M. Fiore, "Daemon: A network intelligence plane for 6g networks," in *2022 IEEE Globecom Workshops (GC Wkshps)*, 2022, pp. 1341–1346.
[4] M. Gramaglia, M. Camelo, L. Fuentes, J. Ballesteros, G. Baldoni, L. Cominardi, A. Garcia-Saavedra, and M. Fiore, "Network Intelligence for Virtualized RAN Orchestration: The DAEMON Approach," in *2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2022, pp. 482–487.
[5] IBM Corporation, "Autonomic computing white paper – an architectural blueprint for autonomic computing," 2005. [Online]. Available: https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
[6] G. Garcia-Aviles, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, P. Serrano, and A. Banchs, "Nuberu: Reliable ran virtualization in shared platforms," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 749–761.
[7] A. T.-J. Akem, B. Bütün, M. Gucciardo, and M. Fiore, "Henna: Hierarchical machine learning inference in programmable switches," in *Proceedings of the 1st International Workshop on Native Network Intelligence*, ser. NativeNi '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–7. [Online]. Available: https://doi.org/10.1145/3565009.3569520
[8] A. Garcia-Saavedra, J. X. Salvat, D. D. Vleeschauwer, C.-Y. Chang, L. Fuentes, D.-J. Munoz, A. Lutu, M. Gucciardo, M. Fiore, L. E. Chatzieleftheriou, N. Slamnik-Kriještorac, P. Soto, M. Camelo, M. Gramaglia, G. Garcia-Aviles, E. Municio, and N. Mhaisen, "DAEMON Deliverable 3.2: Refined Design of Real- Time Control and VNF Intelligence Mechanisms," Nov. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7525876
[9] L. Fuentes, M. Amor, Ángel Cañete, M. Fiore, S. Alcalá, S. Barmpounakis, I. Chondroulis, I. Belikaidis, E. Kosmatos, A. G. Saavedra, J. X. Salvat, M. Camelo, P. Soto, A. Lutu, G. Iosifidis, D. D. Vleeschauwer, C.-Y. Chang, and A. Pentelas, "DAEMON Deliverable 4.2: Refined design of intelligent orchestration and management mechanisms," Nov. 2022, This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 101017109. [Online]. Available: https://doi.org/10.5281/zenodo.7544155
[10] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. J. Alcaraz, "Vrain: A deep learning approach tailoring computing and radio resources in virtualized rans," in *The 25th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3300061.3345431
[11] W. Xia, S. Rangan, M. Mezzavilla, A. Lozano, G. Geraci, V. Semkin, and G. Loianno, "Generative neural network channel modeling for millimeter-wave uav communication," *IEEE Transactions on Wireless Communications*, vol. 21, no. 11, pp. 9417–9431, 2022.
[12] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (mlops): Overview, definition, and architecture," 2022. [Online]. Available: https://arxiv.org/abs/2205.02302
[13] "Kubernetes." [Online]. Available: https://kubernetes.io/
[14] "Kubeflow." [Online]. Available: https://www.kubeflow.org/
[15] "Eclipse zenoh." [Online]. Available: https://zenoh.io/