Computer Modeling in Engineering & Sciences

Tech Science Press

Doi:10.32604/cmes.2025.072607

ARTICLE

Multi-Domain Network Intent Policy Enforcement

Ana Hermosilla[1,2,*], **Pedro Martinez-Julia**[3], **Diego R. Lopez**[4] **and Antonio F. Skarmeta**[1,2]

[1]Information and Communications Engineering Department, University of Murcia, Murcia, 30100, Spain
[2]Research & Development Department, Odin Solutions, Murcia, 30007, Spain
[3]Network Architecture Laboratory, Network Research Institute, National Institute of Information and Communications Technology, Tokyo, 184-8795, Japan
[4]Research & Development Department, Telefonica I+D, Madrid, 28050, Spain

*Corresponding Author: Ana Hermosilla. Email: ana.hermosilla@um.es

Received: 30 August 2025; Accepted: 04 November 2025; Published: 23 December 2025

ABSTRACT: In this study, we analyzed the processes involved in the resolution and enforcement of multi-domain network intent policies for intent-based networking (IBN). Previous studies on IBN analyzed the basis of the network intent resolution processes. These processes produce the artifacts required by network intent policy enforcement. Thus, we continued such studies with the inclusion of network intent policy enforcement in the analysis, for which we constructed a model that predicts the accuracy of a multi-domain network intent policy enforcement system. We validated the model by designing a new multi-domain network intent policy enforcement system, and evaluated the accuracy and performance of the new system through experimentation over a large-scale multi-domain platform that involves sites separated by more than ten thousand kilometers. The results show that, on the one hand, the new system improves accuracy by 10% and, on the other hand, that policies obtained from the multi-domain network intents, including the most complex ones, can be enforced in less than 1.75 s in a platform comprising sites located in almost opposite sides of the world. The experiment confirmed that the long distance existing between the sites involved in our experimental multi-domain IBN platform had little impact on the performance of the new system, and that the predictions obtained with the new model are as much as 99% accurate with respect to the behavior observed in the experiment.

KEYWORDS: Intent-based networking; policy enforcement; multi-domain network intents

1 Introduction

Intent-based networking (IBN)[1] enables network tenants to use a high-level and/or formal language to specify the requirements of the network services they want to construct and manage. Such specifications are called network intents [1]. The requirements specified in network intents are generally intertwined with each other. They can be used to determine the components needed to construct the intended network service and the policies that determine their operational boundaries.

Network function virtualization (NFV) platforms, such as Open Source MANO (OSM) [2], can construct a network service from a provided network service description (NSD). Therefore, network intents must be translated into NSDs for them to be deployed in NFV platforms. Similarly, network intents must also be translated into the policies that determine the operational boundaries of the related network services.

[1]Table 1 gathers the definitions of the acronyms we use throughout this document. Table 2 gathers the definitions of the key concepts used throughout this document.

Such policies must be represented in a structure that can be understood by the IBN system that manages the related network service.

Common formats for representing network intent policies in a structured manner are the object constraint language (OCL) [3] and the semantic web rule language (SWRL) [4]. The former is related to the Unified Modeling Language (UML) [5]. The latter is related to the resource description framework (RDF) [6]. The resulting policy definition structures are meant to be communicated to the IBN systems through YANG (yet another next generation) models [7] containing fields conforming to the simplified use of policy abstractions (SUPA) specification [8]. The IBN systems must ensure that the policies are enforced in the network services.

Table 1: Acronyms

Acronym	Definition
CCL	Closed-control loop—formed by the MAPE activities
IBN	Intent-based networking
LLM	Large language model
MAPE	Monitor, analyze, plan, execute
MAS	Multi-agent system
NFV	Network function virtualization
NLP	Natural language processing
NSD	Network service description [2]
OCL	Object constraint language [3]
RDF	Resource description framework [6]
SOA	Service oriented architecture
SUPA	Simplified use of policy abstractions [8]
SWRL	Semantic web rule language [4]
UML	Unified modeling language [5]
VP	State variable and/or configuration parameter

Table 2: Glossary

Concept	Definition
Configuration parameter	Parameter that determines the operation of a function (e.g., number of reserved CPUs)
Network intent	Set of requirements written in high-level language (even natural language)
Policy	Abstract definition of a condition that must hold
Rule	Formal definition of a condition that must hold
State variable	Monitoring metric that defines some state of the system (e.g., CPU load, bandwidth use)

For IBN systems to enforce the conditions set by network intent policies, they must perform a series of non-trivial operations, resulting in disparate results [9]. In this study, we analyzed the state-of-the-art on network intent policy enforcement, as a continuation of previous studies that focused on network intent policy resolution. We identified, on the one hand, the processes and artifacts required to properly enforce

the policies specified in multi-domain network intents, and, on the other hand, the limitations of current network intent policy enforcement systems when applied to multi-domain network intents.

To support our analysis, we formulated a model that maps each system for network intent policy enforcement to a metric that represents the ratio of policy items correctly enforced over the total number of policies specified in network intents, namely the accuracy of the policy enforcement system. The model input is a vector of system descriptors obtained from the design of the system. The model output is the ratio of policy items correctly enforced over the total number of policy items specified in network intents.

Using the model to analyze related works, we found the limitations of the methodologies they used to design their policy enforcement systems, whose accuracy was suboptimal. We then inverted the model using a solver to find out a set of descriptor values to design a system that overcomes those limitations. We constructed a system that enforces multi-domain network intent policies according to those descriptor values. We evaluated the system experimentally to demonstrate that it improves accuracy.

The main contributions of this study to advance the state-of-the-art in network intent policy enforcement are confirmed by the model and system. Particularly, the new system for policy enforcement in IBN increases the accuracy by 10%, reaching 99% of overall accuracy, while keeping its performance on par—taking less than 1.75 s to enforce even the most complex network intents. The factor that most influences the results is how the closed-control loop (CCL) [10] is used by each solution. For instance, most solutions only incorporated one CCL with none or one of them having all activities of a monitor, analyze, plan, execute (MAPE) CCL. However, the model determined that a high number of MAPE CCLs is preferred to avoid enforcement errors. We demonstrated this quality in the evaluation of our solution, as we will detail in Section 5.

The remainder of this paper is organized as follows. First, in Section 2 we will contextualize our study and discuss related work. Then, in Section 3 we will discuss the model we constructed, in Section 4 we will discuss the system we constructed, and in Section 5 we will discuss how we evaluated it to demonstrate its properties and benefits. Finally, in Section 6, we will discuss our conclusion and introduce some indications for future work.

2 Background

IBN enables the construction and management of network services from specifications formulated in a high-level formal language or even in a natural language. Achieving correct, complete, and efficient policy enforcement is an essential requirement for the realization of IBN. The accuracy of network intent policy enforcement systems has a high impact on IBN correctness and motivates the problem statement detailed below.

2.1 Problem Statement

In this study, we investigated the effect that the structure of multi-domain network intent policy enforcement systems has on the accuracy of their enforcement operations. Our goals were:

- Formulating a model that maps the system descriptor values of a multi-domain network intent policy enforcement system to the ratio of policy items correctly enforced over the total number of policies specified in network intents.
- Designing a system that maximizes the accuracy of network intent policy enforcement—the descriptor values used to design the system will maximize the outputs of the model.

2.2 Related Work

The most widespread methods for policy enforcement rely on closed-loop control functions (CCLs) [10]. One of those methods is proposed within the framework CLARA [9] and the IDC solution [11]. On the one hand, CLARA proposes to build a service management model from the input network intents. The model is then fed into a CCL that chooses the actions required to enforce the policies. On the other hand, the IDC solution proposes to decompose intents into several policies represented using logic structures. It enforces the policies using the latest state-of-the-art AI mechanisms on a hierarchical structure of CCLs. In our work, we acknowledged the benefits of the policy decomposition strategy used by both solutions by applying a hybrid of them to our solution. However, since policy relations are not hierarchical and may generate conflicts among concurrent control loops [12], they cannot be properly enforced using a hierarchy of CCLs.

With focus on policies, we found PRD [13]. It is a method for decomposing policies into low-level rules formalized in UML [5] diagrams. It shows that formalizing policies using a logic language allows them to be translated into almost all enforcement systems with a high degree of coverage. Our proposal uses a similar decomposition method and also uses a logic representation. On the other hand, NSL [14] proposes to represent policies as slices. It defines a life-cycle management for network slices that interprets input intents to find which slice they are related to. In [15], the authors take a similar approach to achieve service function chaining for 6G over zero-touch networks. In this case, policies encoded in input intents are mapped to predefined structures. The main advantage of these solutions is their simplicity and reduced cost of implementation. However, such solutions are too strict and inflexible to accommodate all the concepts that can be present in current network intents, as specified in [1].

In line with the zero-touch management proposed by NSL, the work presented in [16] proposes the application of automated machine learning as part of a zero-touch network and service management architecture [10] in the context of 5G networks. Other machine learning methods are suboptimal to converge towards a coherent policy for IBN. In our work, we opted to follow a similar approach but constrained it through the application of category theory, so that the results are even closer to optimum representation, and fewer policy rules are left out, as we will show in Section 5.5.

To tackle the complexity of concepts and other elements contained in network intents, some solutions rely on the use of semantic technologies. On the one hand, Onto-Planner [17] proposes a semantic model based on OWL [18], and uses it in the application of a deep learning method. It shows that the application of logical reasoning to knowledge graphs favors the detection of policy violations. It also shows that the definition of policies in logical and semantic terms enables policy refinement. These results support the use of semantic technologies as a foundation for policy enforcement and verification in IBN. On the other hand, INDIRA [19] proposes an UML schema [5] and an RDF [6] ontology for intent representation. It uses graph theory to identify conflicts, check rules and policies. INDIRA gets intents represented in UML/RDF and implements them into the network. For the time being, INDIRA intents can only use specific/strict terms, so it lacks the appropriate level of abstraction for fulfilling long-term IBN goals.

When multiple intents are deployed in the same system, the intent translation mechanisms must ensure that enforced functions and rules have no conflict. Several solutions target this problem, such as [12,20]. On the one hand, some solutions propose to introduce semantic representation of both network intent and its implementation counterparts. The graphs that represent the latter are compared to find possible conflicts. On the other hand, other solutions propose to represent the intents as bargaining requests and use a bargaining problem solver to find the equilibrium. In our study, we considered both aspects by representing the intents using RDF graphs [6], defining a repeated leather follower game [21], and using a MAS to resolve the game.

Some solutions propose to address the complexity of network intents through the use of declarative programming environments and reasoning engines, such as Prolog [22] and Haskell [23]. They propose to transform the declarative expressions that form a network intent into more formal, but also declarative, expressions formatted in a declarative language like Prolog. Once all concepts are represented as Prolog declarations, the reasoning engine can automatically resolve the required computations for obtaining the descriptions required to deploy the network intents, as well as enforcing and/or validating their policies. In our work, we extended these concepts with high-level and abstract declarations that support functional language and ensure that the category theory axioms hold.

In addition to verifying that the policies defined in deployed intents hold, the behavior of running network services must be checked. Many systems ensure that those network services comply with the widely known serve-level agreements, such as the works discussed in [24,25]. A common denominator in such proposals is the application of the digital twin concept to the network, constructing the so-called network digital twin [26,27]. They propose to configure the network digital twins according to the network intents and analyze the result to determine if it is possible to deploy such intents. Our solution borrows from these solutions the concept of using KPIs, in the form of key-value indicators, to construct a structure that represents the state of the network and that can be compared with the policies.

As exposed by Hexa-X-II [28], new methodologies are required to design IBN systems, including methodologies for properly designing policy enforcement processes. Hexa-X-II promotes methodologies that rely on trust-as-a-service (TaaS) models [29] and enforcing security and privacy from the basement of the architecture. This impacts how intents are translated and how their policies are first represented and later enforced. Hexa-X-II promotes the use of multi-agent systems for resolving policies and orchestrating the services according to them. We acknowledged this proposal and incorporated a multi-agent system into the model and system we proposed.

In summary, related work denotes little to no agreement on how policies are represented—rules, knowledge graphs, declarative language expressions—or how they are enforced. The knowledge obtained by some solution cannot be ported to other solutions. They are conceptually incompatible. An abstract point of confluence is required to overcome this limitation and ensure the quality and interoperability of proposals, both at the conceptual level and the design and implementation levels.

2.3 State-of-the-Art Analysis

We analyzed the works presented in Section 2.2 and identified the most essential steps required to properly realize multi-domain network intent policy enforcement. They are:

1. Producing a set of simple but specific statements in a natural or formal language that represent the policies specified in the given network intent expressed in a high-level language (formal or natural).
2. Producing the material required to realize the network service from the set of statements produced in the previous step:

- A set of semantic rules expressed using OCL [3] or SWRL [4] in relation to a particular knowledge base.
- A set of NSD documents [2] describing the virtual elements that must be instantiated.

3. Producing, for each semantic rule in the set of semantic rules produced in the previous step, the following material:

- A set of state variables $\{y_t^1, y_t^2, \ldots, y_t^n\}$ that must be monitored;
- A goal function that maps the state variables to the level of policy compliance $(h\ \{y_t^1, y_t^2, \ldots, y_t^n\}) \in (0 \ldots 1)$;

- A set of configuration parameters $\{x_t^1, x_t^2, \ldots, x_t^n\}$ that can be changed to ultimately change the state variables; and
- The map that relates the configuration parameters and the state variables $\{y_t^1, y_t^2, \ldots, y_t^n\} = (f\ \{x_t^1, x_t^2, \ldots, x_t^n\})$.

4. Computing, for each semantic rule, the result of the produced expressions to check if the system is compliant with the policies specified in the network intent—namely, that the expressions hold.

2.4 Limitations for Full IBN Realization

Current state-of-the-art systems lack several aspects of the procedure described above required for IBN. Particularly, they lack the following elements:

- A mechanism for the synchronization of enforcement operations in multi-domain scenarios.
- A formal and canonical procedure for getting the set of autonomic controllers required to enforce the policies defined in a network intent.
- A mechanism that transforms the policy-related concepts present in network intents into structures that are suited for policy enforcement operations.
- A mechanism that finds the configuration parameters that can be changed to influence a particular set of state variables.
- A mechanism for checking that the current state of a system is compliant with the policies defined in a network intent.

In addition to IBN requirements, as stated in [9], current systems lack some IBN features that are essential to realize the broader zero-touch network and service management architecture. In summary, those features are:

- Enabling the use of network intents to express user and business requirements.
- Being able to construct high-level declarative policies from those network intents.
- Incorporating a CCL to enforce policies.
- Making use of machine intelligence in decision and resource optimization processes.
- Supporting network services to be constructed using functions deployed in multiple domains, namely, supporting multi-domain network services.

Moreover, management systems must be fully automated, provide quantifiable and predictable service quality assurance, and make use of advanced intelligence to incorporate knowledge that supports obtaining suitable decisions and root cause analysis. However, achieving such functions together in an automation system is still an open research problem [23]. Additionally, there was no proper model that could be used to determine the level of compliance of a mechanism used for policy enforcement.

2.5 Summary

In this section, we have discussed the problem we aimed at, the previous works that aimed at a similar problem, and why they are not suitable for cover the particular requirements of the problem we addressed in our study.

3 Multi-Domain Network Intent Policy Enforcement System Accuracy Model

Once the qualities and limitations of related work were identified, we proceeded to construct the model that predicts the accuracy of a multi-domain network intent policy enforcement system. The model input is a vector of system descriptors. The descriptor values are obtained from the design of the system. We will

detail them below. The model output is the ratio of policy items correctly enforced over the total number of policy items specified in network intents. This model addresses the first part of the problem introduced in Section 2.1.

3.1 Model Formulations

We produced two formulations that correlate the descriptor values used in the design of the policy enforcement system—d_i—and the enforcement accuracy ratio—e_l and e_p. Both formulations exponentiate the descriptor values d_i to a coefficient $c_{i,2}$, multiply them by a weight $c_{i,1}$, and displace them by an origin value $c_{i,3}$.

The two formulations differ in that, in the formulation for e_l, the composition is a linear composition, while in the formulation for e_p, the composition is a product composition. They are formalized in the following equations:

$$e_l = \sum_{i=1}^{n} \frac{1}{n} c_{i,1} d_i^{c_{i,2}} - c_{i,3}$$

$$e_p = \prod_{i=1}^{n} \frac{1}{n} c_{i,1} d_i^{c_{i,2}} - c_{i,3}$$

3.2 System Descriptors

The descriptors we used to formulate the model are directly related to the overall structure of the system. The main procedures of an enforcement system, as discussed in [28], realize the steps introduced in Section 2.3. The procedures are related to the components and interfaces specified in a reference system for network intent translation [30].

When a network intent enters the system, it is sent to the component called the intent manager. This component manages the life cycle of the intent. It sends the input intent to the intent translation engine (ITE), as defined in [30]. The ITE then makes use of the latest technologies in language processing to reformulate a network intent as a policy set, which is a set of self-contained statements derived from those statements that refer to policy in the network intent, in contrast to those referring to the functions and components which form part of the network service definition [5].

After that, another procedure of the ITE translates the policy set represented in natural language to a policy set represented in SWRL [4]. The procedure uses natural language processing (NLP) techniques to generalize the language structures as rule expressions. Then, the procedure binds the resulting expressions to the system that is making the translation by using the specific terms present in the knowledge base of such a system to replace the general terms present in the natural language expressions.

The intent manager is equipped with a procedure that learns the mapping between a set of state variables and the set of configuration parameters (VPs) that can be modified to change the variables. Finally, the policy verifier operates alongside the ITE to retrieve the enforced policies and, after collecting the VPs involved in the policies when needed, assess if the rules included in the policies hold.

Following this process, we identified the logical components involved in the intent enforcement process in their different levels. They are shown in Table 3.

The first and second descriptors are obtained from the logical representation of the intent rules. It is common to add a layer to the decision tree for each variable of the rules [9]. It is also common to have as many policy rules as possible. Instead of processing a big monolithic rule, common solutions process small individual rules [11].

Table 3: System descriptors

i	Descriptor Name
1	Depth of decision tree
2	Number of policy rules
3	Number of CCLs
4	Number of MAPE CCLs
5	Number of measured variables
6	Number of tuned parameters
7	Number of dependent VPs
8	Number of independent VPs
9	Size of stored state
10	Number of non-trivial loops
11	Average number of links per module

The third and fourth descriptors are determined by design decisions. Some solutions only use one CCL [9], whereas other solutions use many [5]. Not all CCLs have all MAPE processes. We identified those CCLs that are full MAPE and those that are not.

The value of the fifth descriptor is obtained from the inputs of the CCLs and the policy rules. The works presented in [24,25] propose to use all possible variables that can be obtained from the underlying controllers. For instance, as we will show in Section 5, a typical platform can provide 28 variables related to a network service that has been instantiated from a network intent.

The value of the sixth descriptor is obtained from the decision and enforcement elements used in the system. Common proposals, such as Hexa-X-II [28], target as few parameters as two to three. Others, such as our proposal, which we will present in Section 4.5, extend such number to 8. Depending on the possibilities of the underlying system, this number can be even bigger.

The seventh and eighth descriptors are obtained by first joining the values of the fifth and sixth descriptors, then identifying dependencies among them, and finally counting the VPs that depend on other VP—set for the seventh descriptor, and the VPs that do not depend on any other VP—set for the eighth descriptor.

The ninth descriptor is obtained by measuring the size of the structure used to store the state information, e.g., a database on disk or memory. For theoretical evaluations, this is obtained by computing the minimum amount of data that would represent the needed storage. For instance, CLARA [9] uses a pseudo-stateless approach, with minimum stored memory, whereas Hexa-X-II [28] uses a database that requires three times as much information. To homogenize the solution space for this descriptor, we represented its values using memory units. As we will discuss in Section 5, we implemented our system with totally stateless components, so that it used a minimum amount of memory, like CLARA.

The tenth descriptor is obtained by analyzing the algorithm used by the system. Trivial loops are those that iterate over a collection or over a strongly bound vector subspace—e.g., an integer range. Trivial loops can be matched to well-known mathematical series. Non-trivial loops have complex conditionals, so they cannot be matched to well-known mathematical series. The values obtained for this descriptor are the result of counting such kind of loops in the algorithms used by the enforcement systems. For instance,

Onto-Planner [17] algorithm has three non-trivial loops, so it cannot be mapped to a functional map of a well-known mathematical series. The algorithms of other solutions, such as CLARA [9], use only one non-trivial loop. We designed our solution without using any non-trivial loop.

Finally, the eleventh descriptor is obtained by analyzing the structure of the proposed system using its component view, which shows the modules that compose the system and their relations forming a graph. The value of the descriptor is obtained by averaging the number of edges of each node of the graph. Most proposals show an average of four or five edges per node. We designed our solution with an average of three edges per node.

3.3 System Descriptor Values

Once the system descriptors that were going to form part of the model were determined, we determined their values for three solutions. The first solution—baseline—represents the most typical solutions, which generally use a single CCL with no MAPE for all VPs. The second solution—reference—represents the best system among the related work, which also has a single CCL for all VPs, but this solution includes MAPE. The third solution—proposal—represents the system we constructed in this study. On it, we used a CCL for groups of VPs. All resulting CCLs were interconnected, forming a graph. The resulting values are shown in Table 4. The baseline system is an approximation to, and represents, how IDC and INDIRA behave, while the reference system is an approximation to, and represents, how CLARA behaves.

Table 4: System descriptors values

i	Descriptor Name	Baseline	Reference	Proposal
1	Depth of the decision tree	3	1	1
2	Number of policy rules	9	3	9
3	Number of CCLs	1	1	8
4	Number of MAPE CCLs	0	1	8
5	Number of measured variables	28	28	28
6	Number of tuned parameters	2	3	8
7	Number of dependent VPs	26	13	8
8	Number of independent VPs	2	13	20
9	Size of stored state	3	1	1
10	Number of non-trivial loops	3	1	0
11	Average number of links per module	5	4	3

3.4 Coefficient Estimation

Once the system descriptors and their values for the involved systems were determined, we replaced them in the equations of the model and computed the model coefficients $c_{i,j}$ using *fsolve* from *SciPy Optimize*, the well-known *Python* library for scientific computation. We set *fsolve* with the objective of minimizing the following vector function:

$$f(C) = \left\{ \left(\sum_{i=1}^{n} \frac{1}{n} c_{i,1} d_{j,i}^{c_{i,2}} - c_{i,3} \right) - a_j \right\}$$

In it, C is the matrix of coefficients $c_{i,k}$, $d_{j,i}$ is the descriptor i of system j, a_j is the accuracy of system j, $\{\ldots\}$ is the vector of residuals that *fsolve* minimizes. The resulting coefficients are shown in Table 5.

Table 5: Coefficients

i	$c_{i,1}$	$c_{i,2}$	$c_{i,3}$
1	0.08	−1.17	−0.88
2	−0.66	−69.30	−0.93
3	−0.43	−15.61	−0.99
4	0.00	1.82	−0.90
5	0.68	0.07	−0.07
6	0.00	3.40	−0.90
7	241,425.25	−7.08	−0.90
8	−1.33	−0.02	−2.20
9	0.04	−8.70	−0.92
10	−2.13	0.00	−3.03
11	1235.33	−8.59	−0.90

3.5 Ideal Optimal Solutions

From the set of coefficients $c_{i,j}$, we determine two optimal solutions, *Optimal-1* and *Optimal-2*, which are obtained by resolving the equations for maximizing enforcement accuracy. *Optimal-1* reflects rounded and realistic values, whereas *Optimal-2* reflects the direct outputs obtained by the equation resolving algorithms. The results are shown in Table 6. The theoretical optimality of these solutions is warranted by the resolution of the model introduced in Section 3.1. To demonstrate the relation between the theoretical optimality and real (experimental) optimality, we carried out the evaluation experiments, which we will detail in Section 5. In them, we found the close relation between the theoretical and experimental results, evidencing the validity of the model and optimality of the systems designed according to the mentioned descriptors.

Table 6: System descriptors values for the optimal solutions

i	Descriptor Name	Optimal-1	Optimal-2
1	Depth of the decision tree	1	1
2	Number of policy rules	2	2
3	Number of CCLs	8	6
4	Number of MAPE CCLs	8	8
5	Number of measured variables	80	83
6	Number of tuned parameters	8	8
7	Number of dependent VPs	8	8
8	Number of independent VPs	80	88
9	Size of stored state	1	1
10	Number of non-trivial loops	1	1
11	Average number of links per module	3	3

3.6 Category Theory

A key aspect of our proposal is the application of category theory. Category theory is a branch of mathematics that provides mechanisms for describing, formulating, and formalizing mathematical structures using particular objects and relations between them (see [31,32]). The fundamental concepts of category theory enable the structures to be analyzed through these relations without having to assess the objects.

By definition [31], a category $\mathcal{C}$ has the following constituents:

- A collection $\mathbf{Ob}(\mathcal{C})$ of objects that are arbitrary elements, such as sets or semantic atoms.
- For every two objects $c, d \in \mathbf{Ob}(\mathcal{C})$, a set $\mathcal{C}(c, d)$ of morphisms from c to d exists—also defined as $f : c \to d$, which are relations, maps, etc., from c to d.
- For every object $c \in \mathbf{Ob}(\mathcal{C})$, we have an identity morphism on c defined as $\mathbf{id}_c \in \mathcal{C}(c, c)$.
- For every three objects $c, d, e \in \mathbf{Ob}(\mathcal{C})$ as well as morphisms $f \in \mathcal{C}(c, d)$ and $g \in \mathcal{C}(d, e)$, we have the morphism $g \circ f \in \mathcal{C}(c, e)$, which is a composite of f and g.

Furthermore, the aforementioned constituents must satisfy the following conditions:

- Unitality: Composing any morphism $f : c \to d$ with the identity on c from the left or the identity on d from the right does nothing. Thus, $f \circ \mathbf{id}_c = f$ and $\mathbf{id}_d \circ f = f$.
- Associativity: For every three morphisms $f : a \to b$, $g : b \to c$, and $h : c \to d$, we have that $(h \circ g) \circ f = h \circ (g \circ f)$.

Another important concept of category theory is the functor. A functor is a mapping between categories, wherein objects are sent to objects and morphisms to morphisms. Formally, considering categories $\mathcal{C}$ and $\mathcal{D}$, the functor from $\mathcal{C}$ to $\mathcal{D}$ is denoted as $F : \mathcal{C} \to \mathcal{D}$ and has the following conditions:

- For every object $c \in \mathbf{Ob}(\mathcal{C})$, we have the object $F(c) \in \mathbf{Ob}(\mathcal{D})$.
- For every morphism $f : c \to d$ in $\mathcal{C}$, we have the morphism $F(f) : F(c) \to F(d)$ in $\mathcal{D}$.

Similar to the aforementioned constituents, the functor constituents must satisfy the following conditions:

- For every object $c \in \mathbf{Ob}(\mathcal{C})$, we have $F(\mathbf{id}_c) = \mathbf{id}_{F(c)}$.
- For every three objects $c, d, e \in \mathbf{Ob}(\mathcal{C})$ as well as morphisms $f \in \mathcal{C}(c, d)$ and $g \in \mathcal{C}(d, e)$, we have that $F(g \circ f) = F(g) \circ F(f)$ holds in $\mathcal{D}$.

A concept of category theory that is particularly important for our study is the monad. A monad on the category $\mathcal{C}$ is an endofunctor, a functor from a category to itself, and is defined as $T : \mathcal{C} \to \mathcal{C}$. It is accompanied by two natural transformations that are morphisms on the category of functors that map one functor to another. The natural transformations are as follows:

- First, we have $\eta : 1_{\mathcal{C}} \to T$, where $1_{\mathcal{C}}$ is the identity functor on $\mathcal{C}$.
- Second, we have $\mu : T^2 \to T$, where T^2 is the functor $T \circ T : \mathcal{C} \to \mathcal{C}$.

The following conditions must hold for a monad to be correctly defined:

- First, it must hold that $\mu \circ T\mu = \mu \circ \mu T$, where $T\mu$ and μT are obtained using horizontal composition ~ [32]. Here, both sides are $T^3 \to T$.
- Second, it must hold that $\mu \circ T\eta = \mu \circ \eta T = 1_T$, where 1_T is the identity $T \to T$, and the other expressions are also from T to T.

In summary, the simple but powerful concepts defined by category theory allow reasoning on considerably more complex concepts, as we discuss throughout this paper. Recently, category theory has been applied to resolve research problems in computer networks, particularly in network management. For instance, researchers in [23] investigated the application of category theory to support the implementation of an intent-based networking solution. In the following sections, we describe how we applied category theory to construct our model and strengthen our outcomes.

3.7 Overall Theoretical Justification

Once we obtained the descriptors, we formulated the theories that justify the theoretical relation between the system descriptors and policy enforcement. Using such a relation, we formulated a set of recommendations for constructing systems that minimize policy enforcement errors. The formulation of the theorems is subject to the following definitions.

The network service states are represented in the category $\mathcal{C}$, whose objects are arbitrary network service states and whose morphisms are transitions from arbitrary network service states to more stable network service states. The limits of such morphisms are acceptable states. Stability here is measured as the inverse of complexity. A more stable state is a less complex state. The entropy of the information that describes network service states represents its complexity and, therefore, it can be used to determine when a state is less complex than others.

Additionally, intent policies are also categorical [23]. Policies are represented in the category $\mathcal{D}$, whose objects represent policy sets and whose morphisms represent policy modifications.

Categories $\mathcal{C}$ and $\mathcal{D}$ are related as follows:

- A functor maps all objects representing acceptable network states to the object representing the single policy set that verifies them.
- A functor maps all objects representing policy sets to the object representing the single acceptable network state that verifies all policy sets.

By following the parallel arrows, we verified that policies influence the morphisms of the category formed by the acceptable states of a managed network service. The functor defined above maps the morphism from $\mathcal{C}$ to the policy set that represent the motivation for the state change that such morphisms represent.

Considering these results, we obtained a formal representation of the current state of a network service, and remarked that there exists an isomorphism between such a representation and the formal representation of the policies. A mechanism for intent policy enforcement that follows such a relation is consistent and convergent.

In addition, once equipped with all categorical results, we proposed to connect the morphisms in $\mathcal{C}$ as functions in the management software. Therefore, we proposed a slightly constrained functional paradigm to deliver the aforementioned consistency to the network service operation. Our paradigm can be instantiated in any programming language, provided it supports basic function operations—namely, function definition and composition.

Furthermore, we extended the definition of the paradigm with additional equipment to form a set of design principles for network management software. In the era of softwarization, relying on well-known design principles is essential. We gathered those design principles that have been demonstrated to provide the best results. For instance, functional and declarative programming languages are experiencing a new wave of interest. However, switching to a functional or declarative programming language is not needed at all, provided that the language currently used has certain properties. We included those properties as part of our design principles, which we will discuss in Section 4.4. For instance, introspection and meta-language operation are two of them. These are easily found in the top-used programming languages nowadays.

3.8 Theorems

In the following sections, we present the theorems we needed during the present study to ensure the properties required by the target problem.

3.8.1 State Policy Embedding (SPE)

Theorem 1 (SPE): *A policy set cannot be embedded into a network service state. Thus, there is no complete map that takes objects in the category of policy sets to objects in the category of network service states.*

However, there exists a map—more specifically, a functor—that takes objects in the category of network service states to objects in the category of policy sets. Conceptually, a network service state can be embedded into a policy set.

Moreover, there exists another map—also a functor—that takes objects in the category of network intents to objects in the category of policy sets.

Proof: The theorem is proven by accomplishing the following concepts:

- Using a DSL for formulating components and agents.
- Formulating a functor that maps component functions to state morphisms.
- Implementing a search function based on the state monad and different structures, such as the immutable map.

Since those concepts are present in software development guidelines, our theorem holds. □

3.8.2 State Monad and Relating External Elements (SMR)

Theorem 2 (SMR). *Considering the category of objects representing network service states, in which each object is a monad, there exists at least one composite function f that takes S_{t-1} to S_t for each state transition $S_{t-1} \to S_t$.*

$$f : T(S_{t-1} \to S_{t-1}a) \to (a \to T(S_{t-1} \to S_t b)) \to T(S_t \to S_t b)$$

According to such an expression, there exists a morphism in the Kleisly category, denoted by $a \to T(S_{t-1} \to S_t b)$, that involves the external elements a and b in the state transition $S_{t-1} \to S_t$. We identify that these external elements are connected to the prior and ulterior policies and events.

Proof: The proof of this theorem is enclosed in its definition. □

3.8.3 Evolution Over Time (EOT)

When a new policy arrives, the current network service state can become inconsistent because it contradicts the new policy. An inconsistent network service state will also be more complex—there are things to fix that will make the state transition to a consistent and less complex state.

Theorem 3 (EOT). *The EOT theorem states that we can obtain the changes needed to be applied to correct a possible inconsistent network service state by constructing a parallelogram using the following structures:*

- *The map $g : \mathcal{D} \to \mathcal{D}$ that takes the object that represents the previous policy set to he object that represents the new policy set.*
- *The map $f : \mathcal{C} \to \mathcal{C}$ that takes the object that represents the current network service state to the object that represents the target network service state.*
- *The functor that takes:*
  - *the object that represents the current network service state to the object that represents the previous policy set;*
  - *the object that represents the target network service state to the object that represents the new policy set; and*
  - *the map $f \in \mathcal{C}$ to $g \in \mathcal{D}$.*

Following the arrows, we can get the map f which, by the SMR theorem, is linked to the external elements that represent the changes that must be applied to the network service to take it from its current state to the target state.

Proof: The proof of this theorem is enclosed in its definition. □

3.9 Summary

In this section, we have discussed the construction of the model that maps system design descriptors to accuracy and introduced the theoretical tools and new theorems that support the model.

4 Proposed Multi-Domain Network Intent Policy Enforcement System

To address the second part of the problem introduced in Section 2.1, we constructed a system for enforcing policies specified in multi-domain network intents. The system was constructed following the results obtained with the model presented above. However, ensuring the resulting system respects the descriptor values obtained with the model is not trivial. We had to delve in design patterns and principles that supported it.

Below, we will present the required elements in Sections 4.1–4.4. After that, we will present the system we designed in Section 4.5 and how it is integrated into a complete IBN system in Section 4.6.

4.1 Paradigm Shift

Nowadays, network softwarization is widely promoted to break the limitations of current network systems [33]. Despite being highly flexible, software components are, however, much less consistent than their hardware counterparts. Similar to the halting problem in computability theory [34], the behavior of intent-based network systems cannot always be fully predicted or verified a priori. Hence, self-monitoring and closed-loop control mechanisms are required to ensure operational consistency [35].

To improve the consistency of network software, network softwarization processes must use a design and programming paradigm that is particularly tailored to its context and environment. Thus, allowed structures and expressions must be slightly constrained to eliminate most—if not all—undesired effects.

Softwarized modules must be constructed using a paradigm that results from mixing a pure functional paradigm and a declarative paradigm. The benefits are as follows. On the one hand, the functional paradigm is open to many points of view, so it is not limited to languages like Haskell [36]. More general languages, such as Python, can follow the paradigm by just imposing their restrictions. Some verifiers are useful to test the result—e.g., Deal framework for Python [37].

On the other hand, the declarative paradigm is not so well adopted by general programming languages. Some binding mechanism is, however, needed. In our work, we adopted Prolog [38] as the specific declarative programming language, because there are bindings to use Prolog from many languages—e.g., Janus for Python [39].

By considering that the goal of a VNF system is to host network services and applying the concept of IBN to the platform, the VNF system is actually a partial function, and it must be designed as such. It has many mechanisms, but does not have the specific logic that defines the behavior of the network services. Such logic is provided by the network intent. Once a network intent is received and interpreted by the system, the partial function becomes a final function.

This view of the system allows enforcing verifiability qualities, which is a key design principle in our proposal.

4.2 Microdesign Patterns

Service-oriented architecture (SOA) enables structuring large systems as a large set of services that interact with each other to achieve some goal [40]. For instance, we adopted SOA principles to construct the MAS of our multi-domain network intent policy enforcement system. We will detail this in Section 5.3.1.

Component-based systems are used to refine SOA designs. Each service in a SOA is mapped to a large component, which, in turn, is normally split in sub-components. A component is basically an abstract piece of software that has input ports, output ports, and a function—or behavior—that relates outputs and inputs [41].

In order to ensure that IBN systems have the required properties, they must enforce slightly stronger constraints on their components: making sure that the connections between the outputs of some components and the inputs of other components have the same properties as a function composition in functional programming [42]. Thus, a component receives some data through an input port, transforms it according to some function, and sends the resulting data to other component through an output port.

The internal behavior of a component is represented by the map $f : x_1 \dots x_n \to y_1 \dots y_m$, which relates the components inputs and outputs. When two trivial components are connected, they are represented by the composition $g \circ f$. Here, g is the internal function of the component that receives the output of the component whose internal function is f. Imposing the present micro-design on softwarized modules ensures that the consistency of the software modules constructed using the paradigm discussed above scales to the whole system. Recent studies have also explored how category theory can elucidate the internal logic of complex systems such as AI [43], reinforcing its applicability to modeling network intent relations in IBN systems.

4.2.1 Enabling Data Pull

Once there is data available in all input ports of a component, if there are other components connected to the output ports, the component is executed, and the results are sent through the output ports. However, when some data is missing, the component will wait.

By enabling data pull, a component can tell another component to *retrieve* all the input data it needs, *compute* its function, and, finally, provide its output.

Retrieve requests—pull requests—will be forwarded in cascade towards all sources of data. After this, all the components behave normally, since data is provided to the input ports normally.

The key result of enabling data pull is that, although the consistency obtained by the function composition discussed above is retained, the resulting system is more flexible and efficient. Thus, the components of our system can avoid transmitting data without having other components expressively waiting for it.

4.2.2 Applied Theorems

Applying the SMR theorem to the component-based system described in this study means that the data exchanged by the components is based on the state monad. As a result, the management system acquires the following properties:

- Management operations are consistent—they make the state of the managed network service to converge towards less complex and more policy-compliant states.
- The boundaries between implementation, design, and conception are removed: there exists a functor that maps concepts from the conceptual definition of the system to the components in the implementation of the system.

Both properties support the engineering processes required to take high-level requirements for the target system. At the same time, the target system itself gains the ability to take high-level data flows and break them down to the micro-data-flows required to implement the intended functions.

Particularly, in the policy enforcement system, we will detail in Section 4.5 that there exists a functor that takes the high-level concepts of a network intent to the policy sets represented by the objects of category $\mathcal{D}$. At the same time, there exists another functor that takes the policy sets to the CCLs required to implement those goals.

Composing those functors produces another functor that takes the high-level goals encoded in a network intent to the CCLs required to implement them. This is a key outcome of the mechanisms we covered in this study, when applied to the enforcement of policies in IBN systems.

4.3 Cybernetics

The cybernetics research community is one of the major proponents of CCLs [10]. Thus, cybernetics theory is an important source of mechanisms for improving management operations, such as those performed by the network intent policy enforcement systems analyzed in this study. Following basic cybernetics principles, the overall structure of a multi-domain network intent policy enforcement system should rely on closed control loops (CCLs) to monitor and adapt system behavior autonomously, as also observed in autonomic computing frameworks [44], therefore it should have the following properties:

- System goals must be formalized as policy sets. In this context, the policy and rule concepts are synonyms[2].
- For each policy of the set, a CCL node must be defined. Each CCL node must have:
  - a set of monitored VPs,
  - a set of VPs that it can update, and
  - an operation rule specified in some formal language that implements the purposes of the related policy.
- CCL nodes must communicate to each other only through the VPs they share.
- A CCL node must assume that the values of the VPs that it monitors obtained at time t are a function of the values of the VPs that it can update and its operating environment at time $t - 1$.
- Recursivity—and nesting—must be realized by CCL nodes whose actions explicitly engage or dismiss fully operating CCL nodes.
- The system design must take into account that policy priorities are reflected in the resulting graph of CCL nodes. VP dependencies realize those priorities. If the node B depends on a VP provided by node A, the policy implemented by node B has less priority than the policy implemented by node A.
- CCL nodes must follow the properties for the component-based design we will be presented in Section 4.5.
- The system design must take into account that, being the system S the result of composing all the controllers, the purpose (intention) of S is the long-term evolution of the vector of state VPs present in S.
- As a result of the purpose principle, a process must assess that the intention of S is isomorphic to the input intent that governed the construction of S.
- A feedback error function must be defined to obtain the result of comparing the observed evolution of the vector of key state VPs with the desired evolution.

[2] In this study, we considered that policies are abstract concepts, whereas rules are concrete concepts.

4.4 Design Principles

To facilitate the construction of IBN systems that incorporate the paradigm shift, microdesign patterns, and cybernetic qualities discussed above, we formulated the following design principles:

- The goals of each object of the system must be specified using a functional and/or declarative paradigm. It applies to all abstraction levels, from the overall system to specific components.
- The system goals must be connected to the system components through a set of maps, which are similar to the maps we used to define the components that provide intent-to-policy resolution and policy enforcement. Altogether, the resulting sets of maps can take objects from specifications to implementations. Moreover, those are the same maps applied by the system to input network intents—which are subsequently mapped into behavior and specific actions. This resolves the dichotomy between the design and execution maps—making both the same makes the implementation of network intent policies verifiable.
- The components must be stateless. The state must be provided to the components as input. State changes are performed by allowing components to return an updated state.
- Network service states must transition—converge—to less complex network service states. Complexity of an individual network service state is measured by estimating the entropy of the information contained in the structure that represents such network service state—in our system, this is a knowledge graph. More complex information has higher entropy—requires more bits to be coded.
- System properties must be self-verified. Particularly, in relation to the convergence property defined above, the morphisms of the network service management category must verify that their destination objects have less complexity than their source objects. A relaxed principle would accept the condition to be less or equal.
- Feedback mechanisms must cover different levels of control loop types—for each type, there are many loops instantiated: intra-component control loop, system control loops, policy control loops, and network intent control loops. Each loop is intended to verify the system goals at a different level of abstraction. While a component control loop ensures that the functions inside a component operate as expected, the network intent control loops ensure that every network service operates accordingly to the network intents established in the system.
- The functions and/or declarations that cover the goal of the overall system must be theoretically formulated—and empirically verified—as the emergent property of the execution of all components of the system, including operation and management components, as well as their control loops.

4.5 Multi-Domain Network Intent Policy Enforcement System

We followed the principles and the model discussed above to design a policy enforcement system for multi-domain network intents. The construction of the system had two purposes: On the one hand, showcasing the application of the design principles proposed in this study; On the other hand, validating the model that analyzes the quality of the enforcement abilities of the system.

Through the application of the design principles and theorems presented above, we determined that our system must realize the following verifiable functions:

- A function that maps the product between network service states and policy sets to a boolean that indicates if a policy set holds for a network service state.
- A function that maps policy sets to the network states
- A function that maps network intents to policy sets.

We incorporated to our system the components required to realize the functions. First, we specified an autonomic control model, following the cybernetic principles discussed above. The model has the properties required to construct the intent-based self-organized management system. Second, we specified the abstract organization of the components that implement the CCLs, showing which concepts are connected hierarchically and which concepts are connected side-to-side.

Additionally, we designed the system so that the components are instantiated according to the descriptor values obtained with the model, as shown above in Table 4. The key artifacts we needed to construct the system, from high-level to low-level, are the autonomic control model, the organizational diagram, and the CCL that realizes them, which includes the required MAPE activities. We describe them below.

4.5.1 Abstract Autonomic Control Model

In Table 7, we show the descriptors of the controller that implements a CCL for a certain variable—named XX. The design of a controller must include a set of variables to monitor, a goal function to achieve, a set of parameters that can be modified, and one or more assumptions used to learn relations between the variables and parameters.

Table 7: Autonomic control model for controlling the variable XX

Field	Content
Variables	$\{y_t^1, y_t^2, \ldots, y_t^n\}$
Goal	$(h_{\text{CXX}}\ \{y_t^1, y_t^2, \ldots, y_t^n\})$
Parameters	$\{x_t^1, x_t^2, \ldots, x_t^n\}$
Assumption	$\{y_t^1, y_t^2, \ldots, y_t^n\} = (f_{\text{CXX}}\ \{x_t^1, x_t^2, \ldots, x_t^n\})$ Note: f_{CXX} must be learned

The realization of the controller requires the instantiation of MAPE activities. They are formalized in Table 8. The variables and parameters defined above are related to the activity that uses them.

Table 8: MAPE activities for controlling the variable XX

Field	Content
Monitor	$\{y_t^1, y_t^2, \ldots, y_t^n, x_t^1, x_t^2, \ldots, x_t^n\}$
Analyze	Does $(h_{\text{CXX}}\ \{y_t^1, y_t^2, \ldots, y_t^n\})$ hold? Note: New data is used to update learnt f_{CXX}
Plan	$\begin{cases} (h_{\text{CXX}}\ \{y_t^1, y_t^2, \ldots, y_t^n\}) \implies \varnothing \\ \neg(h_{\text{CXX}}\ \{y_t^1, y_t^2, \ldots, y_t^n\})) \implies \\ \quad \implies \text{Find } x_{t+1}^1, x_{t+1}^2, \ldots, x_{t+1}^n \\ \qquad \mid (h_{\text{CXX}}\ f_{\text{CXX}}\ \{x_{t+1}^1, x_{t+1}^2, \ldots, x_{t+1}^n\}) \end{cases}$
Execute	$(\text{Enforce}\ \{x_{t+1}^1, x_{t+1}^2, \ldots, x_{t+1}^n\})$

Following the definitions shown in the tables above, a controller must be defined for each CCL we have in our design. The relations between CCLs are determined by the variables they monitor and/or the parameters they modify. Fig. 1 shows an abstract view of the resulting relations for example variables A, B, and C. This view interacts with the overall design of the system. In our work, this view influenced the construction of the intent manager and determined how policy enforcement elements were instantiated and connected. We will detail them in Section 4.6.

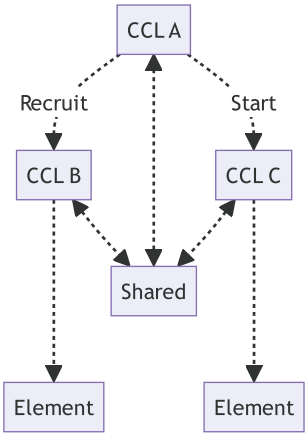

Figure 1: Abstract organization diagram–top level

4.5.2 Policy CCL–MAPE Activities

In our design, we realized the policy CCL [10] through a set of MAPE activities [44]. The MAPE process usually starts with the monitor activity. However, our system begins in the plan activity, because the IBN CCL receives the intent that must be planned before having elements to monitor.

The MAPE activities form a loop in CCL, a shown in Fig. 2. The long-term operation is idempotent regardless of the activity considered as the beginning of a cycle. Each activity is detailed in the following sections.

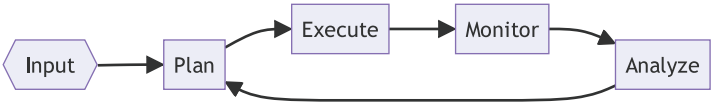

Figure 2: MAPE activities

Plan–Intent to Policies

The goal of the planning activity is to obtain a policy set from a network intent and particularizing it to structures that are understood by the management system.

Obtaining a Policy Set From an Intent The ITE is equipped with a procedure that extracts policy expressions from network intents. It is as follows:

```
def extract_policy_expressions (network_intent):
    phrases = nlp.get_phrases(network_intent)
    policyset = {
        nlp.get_expression(phrase)
        for phrase in phrases
}
return policyset
```

Particularizing a Policy Set The ITE is equipped with a procedure that binds generic policy expressions to particular expressions of the system in which it is instantiated. It is as follows:

```
def particularize_policyset (policyset):
    result_policyset = {
    replace_terms(
        policy,
        {
            term: KnowledgeBase.get(term)
            for term in policy.terms
        }
    )
        for policy in policyset
}
return result_policyset
```

Execute–Policy Enforcement

The execution activity is in charge of enforcing the policies. It hosts the components and functions presented above. The overall process begins when the network tenant inputs a new network intent into the intent manager. It continues by the intent manager, which sends the network intent to the ITE for it to obtain an NSD and a set of policies, and returns them to the intent manager.

Once the intent manager receives the set of policies, it executes a procedure that instantiates as many ARCA nodes as needed to enforce the policies. Each ARCA node is linked to a simple policy expression. ARCA monitors the state variables referenced by the terms included in the policy expression. When needed, it requests the underlying system controllers to change a configuration parameter, which will result in a change in the state variables that ARCA monitors.

Initially, the intent manager only knows the available state variables $y_0 \dots y_n$ and configuration parameters $x_0 \dots x_m$. A function that maps the latter to the former exists as follows:

$$\begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} a_{0,0} & \dots & a_{0,m} \\ \vdots & \ddots & \vdots \\ a_{n,0} & \dots & a_{n,m} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ \vdots \\ x_m \end{bmatrix}$$

The function—represented as the matrix $a_{i,j}$—has the required information to know the configuration parameters that have a higher impact on the set of state variables that the policy enforcer assigns to each ARCA node (possibly only one). Thus, we formulated an algorithm that checks the columns of the matrix $a_{i,j}$ to select those whose value is lower than some threshold. The algorithm then removes the selected columns from the matrix and the corresponding configuration parameters x_j. This reduces the chances that ARCA nodes make to adjust their target variable.

It is worth considering that there exists a matrix $a_{i,j}$ for each moment of time, or each set of measurements of state variables and their corresponding configuration parameters. However, a more generic matrix $a_{i,j}$ that maps a matrix $x_{m,k}$ to other matrix $y_{n,k}$ contains the values that best adjust the map, which are obtained stochastically. Our algorithm finds this matrix and uses the procedure mentioned above to choose the parameters that have a higher impact on the state variables. The matrix system is as follows:

$$\begin{bmatrix} y_{0,0} & \dots & y_{0,k} \\ \vdots & \ddots & \vdots \\ y_{n,0} & \dots & y_{n,k} \end{bmatrix} = \begin{bmatrix} a_{0,0} & \dots & a_{0,m} \\ \vdots & \ddots & \vdots \\ a_{n,0} & \dots & a_{n,m} \end{bmatrix} \cdot \begin{bmatrix} x_{0,0} & \dots & x_{0,k} \\ \vdots & \ddots & \vdots \\ x_{m,0} & \dots & x_{m,k} \end{bmatrix}$$

We formalized our algorithm in the following procedure:

```
y_vec_list, x_vec_list =
       KnowledgeBase.get_state_and_config_vector_lists()
a_mat = NeuralNetwork.learn(y_vec_list, x_vec_list)
col_index_list = select_threshold_columns(a_mat, thershold)
new_a_mat = remove_columns(a_mat, col_index_list)
new_x_vec_list = remove_rows(x_vec_list, col_index_list)
y_vec_names, x_vec_names =
       KnowledgeBase.get_state_and_config_vector_names(
              indices(y_vec_list [0])
          indices(x_vec_list [0])
       )
new_x_vec_names = remove_rows(x_vec_names, col_index_list)
arca_node = create_arca_node()
arca_node.assign_state_variables(y_vec_names)
arca_node.assign_configuration_parameters(new_x_vec_names)
arca_node.assign_map(new_a_mat)
arca_node.run()
```

Once the configuration parameters with the highest impact on state variables are known, the intent manager proceeds to instantiate the ARCA nodes as follows:

```
y_vec_names, x_vec_names =
       KnowledgeBase.get_state_and_config_vector_names(
              indices(y_vec_list [0])
              indices(x_vec_list [0])
       )
new_x_vec_names = remove_rows(x_vec_names, col_index_list)
arca_node = create_arca_node()
arca_node.assign_state_variables(y_vec_names)
arca_node.assign_configuration_parameters(new_x_vec_names)
arca_node.assign_map(new_a_mat)
arca_node.run()
```

It is worth noting that the threshold determines, on the one hand, the precision of the effects of ARCA node decisions and, on the other hand, the amount of work that ARCA nodes will request from the system. This parameter should be adjusted dynamically. Particularly, it should start high and be lowered if the ARCA node change frequency is too high. At the same time, it must be raised if the system gets too loaded (stressed)—meaning that there are too many changes to too many parameters.

Monitor

The monitoring activity involves the retrieval of VPs from the elements that form the network service linked to the specified network intent. Instead of actively pulling for the data, we designed our system to wait for it. VPs are published by the originating elements. Our system subscribes to them, so that it receives the updates whenever they are available.

Publication of Variables and Parameters Each iCPN adapter connected to a monitored element is equipped with a process that implements the following loop:

```
while True:
    before = now()
    for vpi in monitored_vp_list:
        value = element.monitoring_interface.read_vp(vpi)
        name = get_icpn_name(element.name, vpi)
        icpn.put(name, value)
time.sleep(monitoring_period - (now() - before))
```

This procedure takes the VPs obtained from all monitored elements and makes them available as data objects that can be retrieved through iCPN. A name is associated with each data object. The data objects will only be sent upon request to the elements that request them through iCPN and in the moment they request them.

Each data object will be cached in all intermediate network forwarding elements that are in the path between the iCPN adapter that owns the data object and the element that requests the data object. Ulterior requests for the same data object will be responded to by the closest cache to the requester and cached in the network elements that are in the path between the cache that provides the data object and the requester.

Benefits of Using a Cache Hierarchy in the Network The optimum mechanism for data transmission from a single point to multiple points is a system of forwarders organized as a tree. The tree root is the source point of the transmission, and the leaves are the destinations of the transmission. This scheme initially requires all data destinations to receive the data synchronously when it is being sent.

Introducing caches in the middle points of such a tree enables data destinations to receive the data when they are ready to process the data. This transforms the explicit tree described above into a virtual tree. Each time a new destination requests the data, a branch of the tree is instantiated. Thus, only the branches corresponding to destinations that have received the data are instantiated. Moreover, if there is no destination ready when the sender point transmits the information, the tree remains virtual.

When caches reach their limit, old data objects are removed. This way, the instantiated parts of the virtual tree are minimized. Together, these caching functions ensure that the data transmission mechanism in iCPN is totally efficient, dynamic, and scalable.

Naming Data Elements Data names are like universal resource identifiers in which several fields are separated by slashes. The fields are: domain name, tenant name, data type, data source element identifier, data metric identifier, aggregation function applied, compression applied, and version of the data.

Benefits of Using Names Using names enables the unique identification of each data object, which in turn enables data caching and asynchronous data transmission to multiple points. Including hierarchical information in names enables the definition of different data caching and data distribution algorithms. They use any of the fields described above to determine where and when the data will be cached.

Data Element Naming Example

An example entity-variable pair from OpenStack is:

`(5bb6fda3df7d51c6bfec1dd0c5aacc4e, metrics.network.incoming.bytes)`

Which will be translated into the following iCPN name:

ccnx://example.com/tenant-x/telemetry/

```
5bb6fda3df7d51c6bfec1dd0c5aacc4e/
metrics.network.incoming.bytes/current/c-false/0
```

The name specifies the domain (example.com), the tenant (tenant-x), the object type (telemetry), the entity (5bb6fda3df7d51c6bfec1dd0c5aacc4e), the variable (metrics.network.incoming.bytes), the function to apply (current, which refers to a point value—not an aggregated value), the compression to apply (c-false, which means no compression will be applied), and the version of the data (0, which means the latest value).

Analyze–Policy Verifier

The procedures used by the policy verifier are as follows:

```
def check_policies ():
  for pi in policy_set:
    if not verify_policy(pi) and return_on_error:
        return
def verify_policy (p):
    for e in p.expressions:
        res = evaluate_expression(e)
        if not res.ok:
                    icpn.put(get_policy_error_notification(p.name,
res.error))
            return False
    return True
def evaluate_expression (e):
    for operand in e.operands:
        if operand.is_expression:
           res = evaluate_expression(operand.as_expression)
        else:
            res = get_value(operand.name)
        if not res.ok:
           return Result(ok=False, error=res.error)
        operand.value = res.value
    try:
        return Result(
            ok=True,
            value=execute_operation(
                e.operation,
                [operand.value for operand in e.operands]
               )
        )
```

(Continued)

```
        catch:
        return Result(
            ok=False,
            error=f'Expression "{e.name}" failed!'
        )
    def get_value (vp):
        if vp not in own_cache or own_cache[vp].age > AGE_MAX:
            own_cache[vp] = CacheEntry(value=icpn.get(get_icpn_name(vp)))
        return own_cache[vp].value
```

We implemented the function `execute_operation` by calling primitive functions. For instance, to execute an `and` operation, `execute_operation(and, [a, b, c])` in Python is implemented as `return functools.reduce(operator.and_, [a, b, c])`.

The call to `get_value` is executed just when the expression that verifies a policy requires a value. It is stored in a cache owned by the process and reused by the following instructions that require the same data unless they are executed after `AGE_MAX` seconds.

We used caches in this layer to improve overall efficiency. Although a data object that has recently been retrieved will be found with high probability in the cache implemented by the network layer code used by the process that retrieved it, the network layer code runs in its own process and has its own context.

Thus, calls to the network layer code require the intervention of the context manager, which is costly. Storing data objects in a structure owned by the process that uses them—such as the dictionary `own_cache` used in the code above—minimizes the number of calls to the network layer code, and hence increases overall efficiency of the IBN system.

4.6 Integrated IBN System

For the multi-domain network intent policy enforcement system to be fully operational, it requires the other functions present in the intent manager, but also additional functions provided by the underlying platforms and other management components. We thus constructed a complete IBN system, which supports the whole life-cycle of a network intent and its associated network service.

The integrated IBN system includes functions for the reception of a network intent, its translation to an NSD, and a set of policies, the enforcement of the NSD and the policy enforcement (which is the focus of this study), and the monitoring of the VPs. The diagrams shown below depict the data flow of these functions.

4.7 Potential Drawbacks

By following our model, and in comparison to the baseline and reference systems, as shown in Table 4, our system had some particularities that could pose drawbacks in some situations. First, our system managed more policy rules than the reference system, so the elements using the policies used more resources than the reference system.

Second, our system instantiated many more CCLs and MAPE CCLs than the baseline and reference systems. Although the CCLs and MAPE CCLs of our system did not manage as many variables as the baseline and reference systems, they required more system resources to start up. Our system "in fresh" is bigger than the others. It can be a drawback in platforms with constrained resources. Third, our system had to manage more independent variables than the baseline and reference systems. This implies that the measurements were less aggregated, and the processing required more memory to have a view of the system.

We evaluated our approach over a common virtualization platform with typically available components, as discussed in the following section. Our application scenario was, however, limited to exercising the resolution and enforcement of network intents involving multiple requirements.

4.8 Summary

In this section, we have discussed the application of our model to design a new system with better accuracy than previous state-of-the-art systems. It required us to involve results from several areas, such as cybernetics, and gather together a set of design patterns and principles that we had to follow to ensure the intended properties in the enforcement system. We support them to be followed by future systems to achieve the required qualities. Finally, we described the system we designed using the model and design patterns and principles.

5 Evaluation

We evaluated the outcomes of this study through the execution of many intent resolutions and their corresponding policy enforcements with different configurations—one used the baseline solution, another used the reference solution, and another used our solution. We conducted the evaluation as follows. First, we prepared an experimentation platform that had the basic functions required by this study. We will detail it in the next section.

Second, we defined a scenario that represented the goals of the system we constructed. We will describe it in Section 5.2. Third, we implemented the system, as we will describe in Section 5.3. Fourth, we defined and executed an experiment that used such an implementation.

The experiment execution was guided by an experiment manager. We will detail these in Section 5.4. Fifth, we put together the experiment results and applied our model to obtain the accuracy ratio it predicted for each configuration. We will detail this in Sections 5.5 and 5.6.

5.1 Experimentation Platform

We carried on the experiments in a distributed experimentation platform. It involved three domains—namely, the National Institute of Information and Communications Technology (NICT), Telefonica, and the University of Murcia (UMU). Each domain hosted a set of high-performance computing machines.

Specifically, the NICT domain hosted nine computing machines described as *Dell PowerEdge R610 with 2 x Xeon 5670 2.96 GHz (6 core / 12 thread) CPU, 48 GiB RAM, 6 x 146 GiB HD at 10 kRPM, and 4 x 1 GE NIC*. The interconnection network consisted on two physical switches described as *HP ProCurve 1810G-24*. All machines were connected to both switches. One switch was connected to the external network, which is connected to JGN[3], which, in turn, is connected to GÉANT[4].

The remote domains hosted the following elements. On the one hand, both remote sites deployed the virtualization and orchestration infrastructure needed to run the VNF instances used in our experiments,

[3] Japan High Speed R&D Network.
[4] GÉANT Network.

including the OpenStack platform, the local controllers, and the monitoring agents required for policy enforcement and verification. On the other hand, the UMU domain hosted the same elements as the Telefonica domain, with the exception of several proprietary management tools and traffic analysis components that were only available in the Telefonica infrastructure. Regarding the network, UMU is connected to RedIRIS, which is connected to GÉANT. Telefonica domain used the network exchange point in Madrid to access GÉANT. Thus, the global networks involved in our platform were JGN and GÉANT. Both are dedicated to research and academia, and both provide high-speed and low-latency communications.

Over this multi-domain platform, we deployed all components presented in Section 4.5, which formed a multi-domain experimentation scenario, as depicted in Fig. 3. The components were distributed—and replicated when needed—over three physically-separated network domains—namely, NICT, Telefonica, and UMU. All domains used OpenStack as the VIM, which deployed the network functions that composed the network services. All VIMs were connected to the management and policy enforcement solutions analyzed in this study.

The management and operations system that incorporates the components presented in Section 4.5, as detailed in [45], was set to construct a network service that comprised several functions from all involved domains. The network service is derived from a basic one, whose diagram is shown in Fig. 4. Each network service was specified through a network intent. Our system interpreted the network intent, constructed the network service, and enforced the policies specified in the network intent.

As communication in iCPN is based on publish-subscribe, security and privacy policies are enforced through access control rules. This takes advantage of the fact that iCPN uses names to organize the data object space. Those names are structured in hierarchical paths. The path to which the publishing element of a domain responds will have particular access control rules to allow other domains to access the data it publishes.

5.2 Application Scenario

The target scenario of this study consists of the resolution of many network intents, the instantiation of the network services they specify, and the enforcement of the policies they specify. We evaluated our solution in this scenario, so our solution enforced the policies of many different network intents.

Additionally, each execution of our experiment was subject to forced variations of the operating environment, which were particularly conformed to make the network service transition to a state that broke some policies and, therefore, forced the management system to react in order to correct the situation.

One of the network intents we used as input in our evaluation specified a network service that was managed in a way that maximized the level of customer satisfaction by tuning the number of resources—CPU and bandwidth units—allocated for the network service. Following the design process introduced in Section 4.4, we identified the VPs involved in the process, as well as the CCLs needed to enforce the policies the network intent specified.

5.3 Implementation

We implemented a proof-of-concept application to realize the system presented in Section 4.5 and respond to the scenario discussed above. Most of the code was written in Python and Hy [46]. We also made use of Prolog as a reasoning engine—more specifically, SWIPL [47], which we used from the code in Python through the Janus library [39]. We encapsulated the micro-components of our system in libraries and the bigger components in Docker containers.

Through the application of the design principles we discussed in Section 4.4, we took the concepts of our system design to their implementation counterparts. This followed a one-to-one mapping from the conceptual design to the component design and, in turn, the final code deployment. All abstraction levels were formally connected through the corresponding functor, as presented in Sections 3.7 and 3.8.

The result is a set of key components of the experiment and their interactions. The components were those implementing network intent resolution, policy enforcement, policy verification, network service instantiation, and domain interconnection. We will detail them in the following sections.

The component interactions were as follows. First, the network tenant inputted a multi-domain network intent through the user interface of the IBN system. Second, the network intent is partially resolved, which means that only the concepts that can be resolved locally are resolved. Third, an intent formed with the concepts that are not resolved is sent to another domain. Fourth, the concepts covered by the received resolution results are subtracted from the previous concepts. The result forms a network intent that is sent to the other remote domain, from which the final resolution is obtained.

Fifth and final, the local NSD is enforced in OpenStack through OSDM, the VPN is updated to consider the involvement of the remote domains in the new network service, and the policy manager (ARCA manager) is requested to enforce the policies, and the policy verifier is requested to check that the policies hold.

5.3.1 Network Intent Resolution

We implemented the intent translation engine (ITE) as a service that subscribes through iCPN to the appropriate topic used by UI modules to publish new network intents. When the ITE has the NSD and policies that correspond to the received network intent, it publishes them through iCPN. Other services, such as the intent manager, receive the NSD and the policies, and operate to deploy them.

The internal functions of the ITE were implemented using NLP, LLM, and logic reasoning. When a multi-domain network intent is being resolved, the ITE involves the IMDRS. This enables multiple domains to cooperate in the resolution of the network intent. IMDRS is organized as a multi-agent system (MAS). We implemented the internal functions of IMDRS agents using RDFlib and their interfaces using NETCONF over SSH.

In future work, we will make our IMDRS agents able to use NETCONF over iCPN in addition to the currently used NETCONF over SSH. We will also explore mechanisms to get a canonical formal form for a network intent, with the aim of making the translation of equivalent network intents to produce the same expressions.

5.3.2 Policy Enforcement–CCLs

Following the design of the integrated IBN system presented in Section 4.6, we implemented our system to deploy an instance of ARCA for each of the CCLs required to realize the policies present in each input network intent. Each ARCA instance was instantiated as a container. It received via iCPN the information it needed—namely, each instance requested the value of the VPs it required to realize the goals of the CCL it implemented.

Once ARCA determined that a change had to be made to correct a deviation in a VP, it sent a message through iCPN with the details of the change. In our implementation, this message was captured by the resource orchestration service (OSDM-RO) and made effective by sending the appropriate messages to OpenStack—which was the VIM in our implementation.

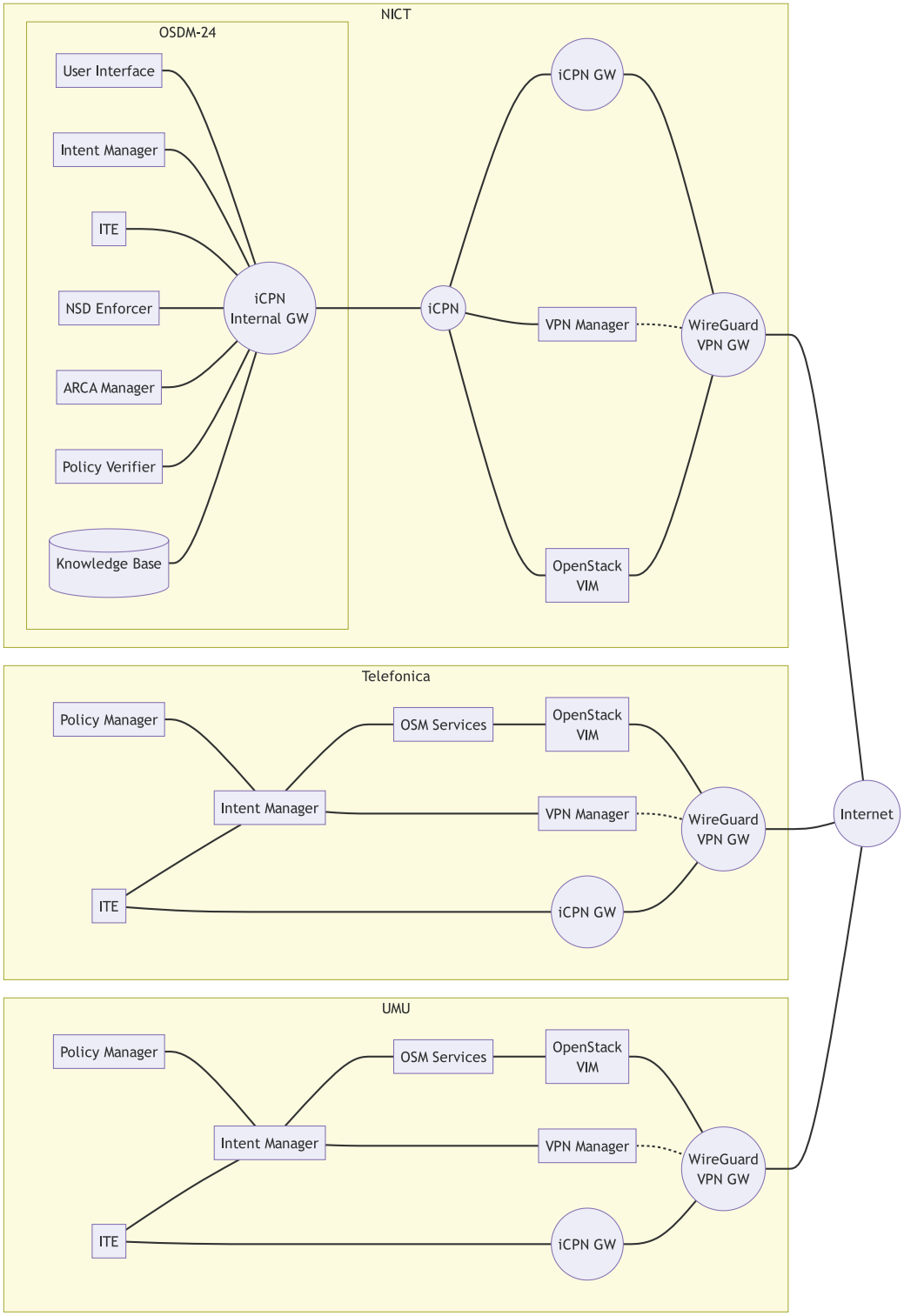

Figure 3: Experimentation scenario

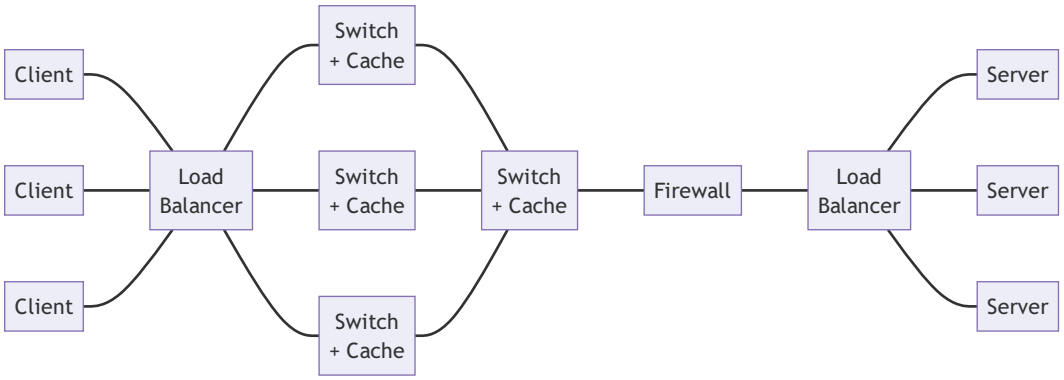

Figure 4: Network service

5.3.3 Policy Verification

We implemented a component for network intent policy verification as described in Section 4.5.2. Its operation was coded as follows. First, an interface was prepared to receive the network intent policies in two different formats. First, we supported policies expressed in SWRL [4] using Turtle [48] as serialization language for SWRL and RDF.

Additionally, we supported policies expressed in OCL 2.4 [3] and encoded in SUPA fields [8] like `supaPolCompConstraint`. In this case, the policies were translated to SWRL-RDF. The contexts they referenced (classes, objects) became RDF atoms—namely, subjects, predicates, or objects. Our implementation included support for YANG as a serialization language. In such a case, a YANG expressions were translated to an RDF expression using YANG2RDF [49].

Once all policies were represented in SWRL-RDF, they were translated into semantic assertions and Prolog predicates. A set of VPs was derived from the expressions. The latest values of the VPs were requested through iCPN. When received, they were translated into semantic assertions, also formulated as Prolog predicates.

5.3.4 Network Service Instantiation

Ultimately, our implementation made use of OpenStack to manage the VNF instances that formed part of network service instances. OpenStack also provided monitoring information from those VNF instances. Both points—the enforcement and monitoring points—were connected as component ports to iCPN. Orchestration components made use of them to create/destroy VNF instances, and CCLs made use of them to retrieve the monitoring data they needed.

5.3.5 Domain Interconnection

Interdomain communication was implemented through VPNs. We connected the elements of the separated domains through two VPNs. One was dedicated to connecting the data plane networks, the other was dedicated to connecting the control plane networks. The VPN dedicated to the data plane connected the OpenStack instances deployed in the separate domains.

The VPN dedicated to the control plane bridged all traffic among the control plane networks of the separated domains. All control plane networks were based on iCPN [50]. The control plane VPN was configured to just bridge the iCPN traffic between domains.

5.3.6 Integration into Operational Environments

Although our system involved many disparate technologies and components, they were structured around the container mechanism. Our system was composed of several containers, defined separately to isolate functions, but they had minimal requirements. Particularly, we used Docker to manage our containers. It is well-known and widely used, and available in typical datacenters. Thus, our system could be easily deployed in most data centers.

Resource-wise, our system did not require a large amount of CPU, but it did produce network traffic. This means that, for environments where the network of the control plane is constrained, additional network bandwidth should be allocated to the control plane before the deployment of our system.

5.4 Experiment Execution

We prepared an experiment manager to instantiate the Docker containers of our implementation and constructed our target application scenario. We executed many configurations that involved the resolution and validation of many network intents with the chosen solutions—namely, the baseline, the reference, and our proposal. The total number of executions was the product of the number of network intents enforced and verified, and the number of solutions.

The execution of a configuration was marked by the following events:

- The system starts.
- A network intent is received.
- The NSD and policy set associated with the network intent are obtained.
- The deployment of the new network service is finished.
- The network intent policies are enforced by the CCLs.

We instructed the experiment manager to record the VPs needed to verify that the policies were correctly enforced. When a policy violation was detected, our experiment manager triggered an alarm that indicated it.

5.5 Experiment Results

Fig. 5 shows the alarms triggered by each solution during the execution of the experiment. The alarms triggered by the baseline solution are homogeneously distributed, whereas the alarms triggered by the reference solution and our proposal are much fewer and sparse.

Fig. 6 shows the enforcement accuracies obtained from the experiments for the solutions evaluated—baseline, reference work, and our proposal. Our proposal reaches 99% of accuracy—two nines—which is over 10% more accurate than the reference and baseline solutions.

Fig. 7 shows the number of concepts present in each network intent. The simplest intent had only two concepts whereas the most complex intent had seventeen. The inner complexity of the concepts was not homogeneous. Thus, some concepts required more work than others to be enforced.

Fig. 8 shows the statistical box plots that represent the distribution of the time used to enforce each multi-domain network intent. The more complex intents required more time to be enforced. The two jumps found around the intents 30 and 110 are because those intents included concepts that were more difficult to enforce than the previous and/or subsequent intents.

Fig. 9 shows the statistical box plots that represent the distribution of the time used to enforce each concept of each network intent. Even though the larger intents had more concepts, they required less time per concept. This demonstrates the scalability of our proposal. The jumps around intent 30 and 110 are still present, although they are softened.

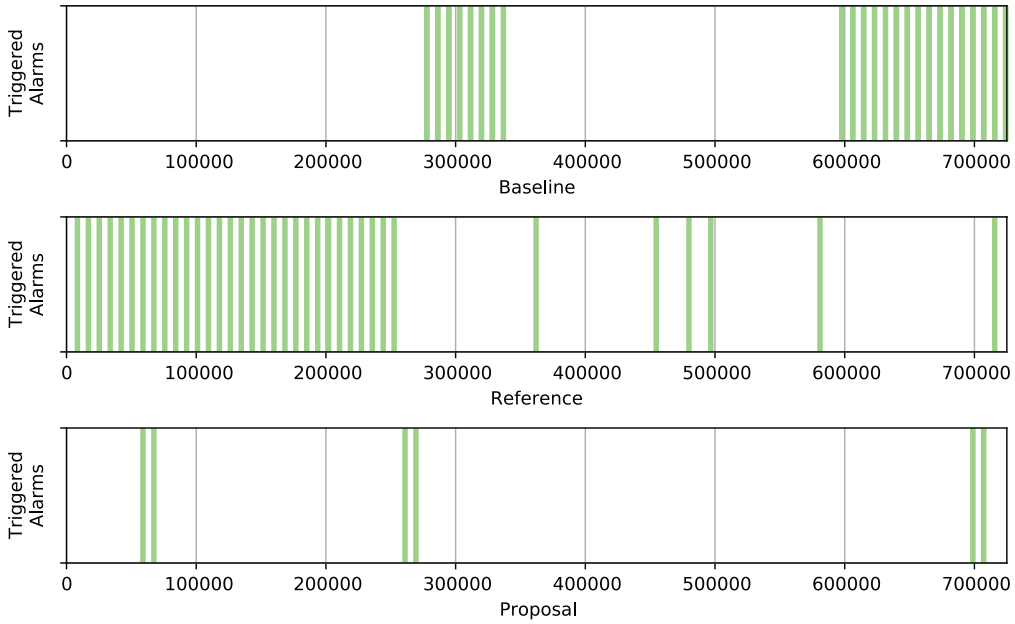

Figure 5: Alarms triggered during experiment execution

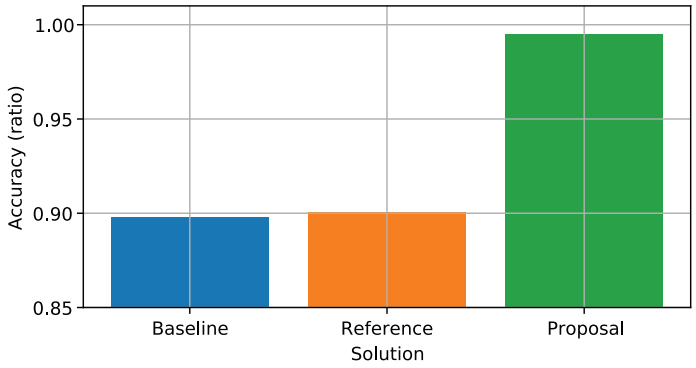

Figure 6: Enforcement accuracies obtained from the experiments

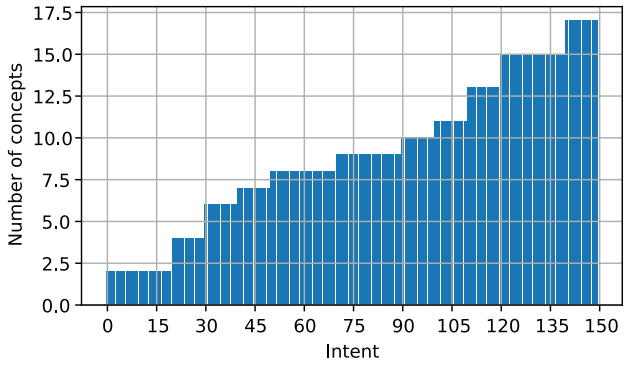

Figure 7: Number of concepts present in each network intent

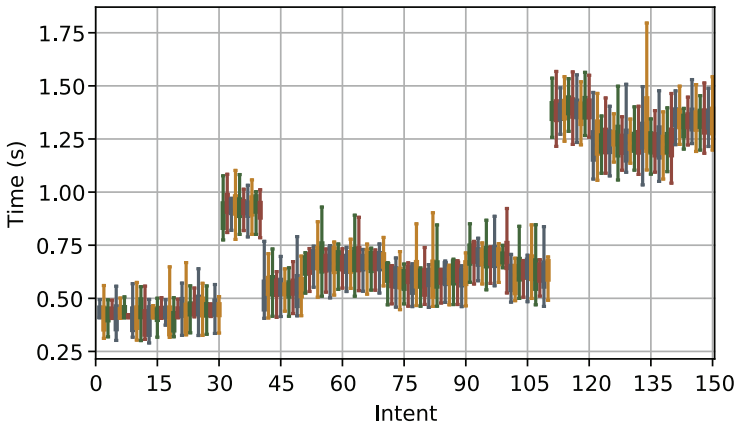

Figure 8: Time used to enforce each multi-domain network intent

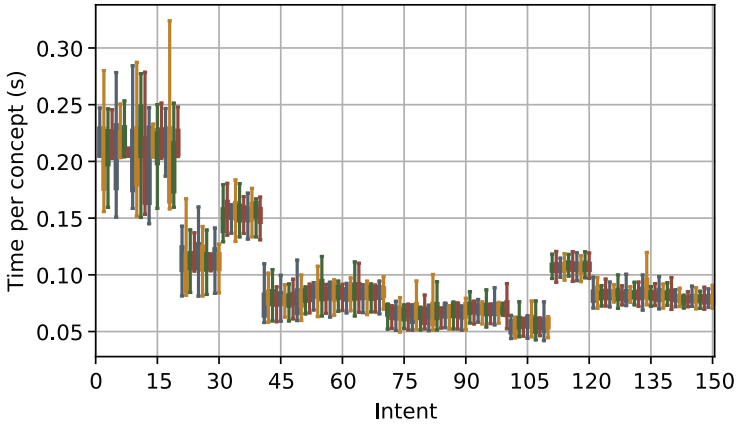

Figure 9: Time used to enforce each concept of each network intent

Fig. 10 shows the statistical box plots that represent the distribution of the traffic used to enforce each multi-domain network intent. It shows a similar pattern to the time used, although the relation between the last and first was not so big. The relation between the time used to enforce more complex intents and the time used to enforce less complex intents is greater than the relation between the traffic used to enforce more complex intents and the traffic used to resolve less complex intents. This demonstrates that the performance of intent enforcement has a higher impact from the complexity of operations than from the number of operations.

Fig. 11 shows the statistical box plots that represent the distribution of the traffic used to enforce each concept of each network intent. It demonstrates that our solution is also scalable in terms of traffic, because the more concepts included in a single intent, the less traffic per concept it uses in the enforcement operation. The behavior complements the demonstration of scalability in time used. Therefore, our solution is scalable and can be applied to very large network intents without incurring much larger enforcement operation times or the use of much larger bandwidth in the control plane.

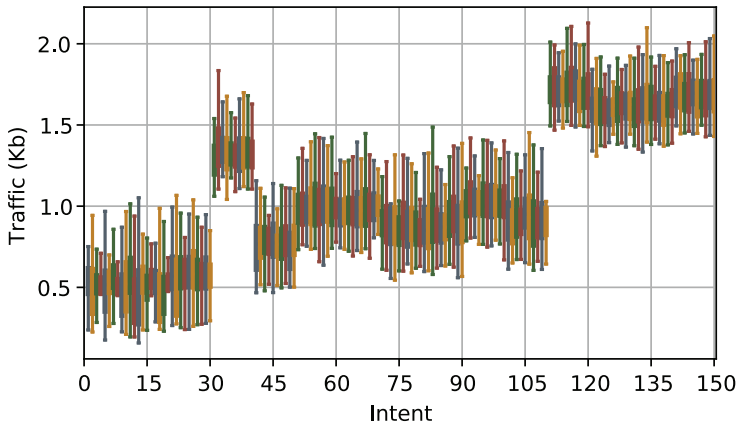

Figure 10: Traffic used to enforce each multi-domain network intent

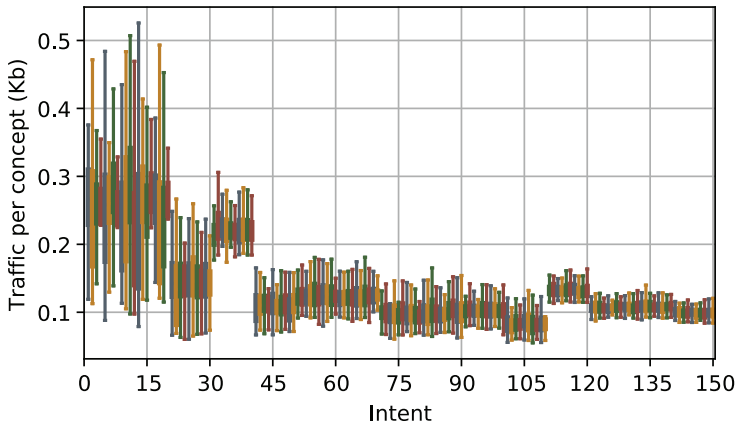

Figure 11: Traffic used to enforce each concept of each network intent

5.6 Model Application

Fig. 12 shows the enforcement accuracies obtained by applying the linear composition model, configured with the descriptor values shown in Table 4. The values are dominated by those obtained from the experiments, as shown in Fig. 6, so it is confirmed that our model does not underestimate the errors. The optimal solutions reach huge levels in the accuracy they achieve, meaning that there is still room to improve the policy enforcement systems. We will explore those possibilities in future work.

Fig. 13 shows the enforcement accuracies obtained by applying the product composition model, configured with the descriptor values shown in Table 4. The values are dominated by those obtained from the experiments, as shown in Fig. 6, and also dominated by the values obtained with the product model represented in Fig. 12. The optimal solutions are able to provide even better results according to this variation of our model.

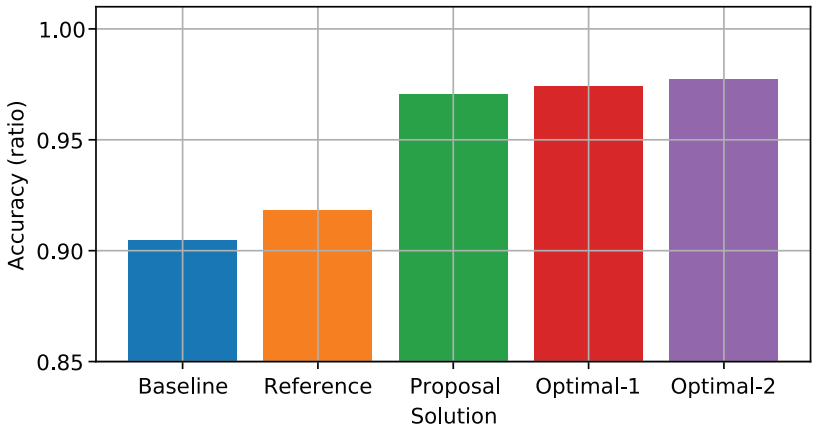

Figure 12: Enforcement accuracies obtained using the model with linear composition

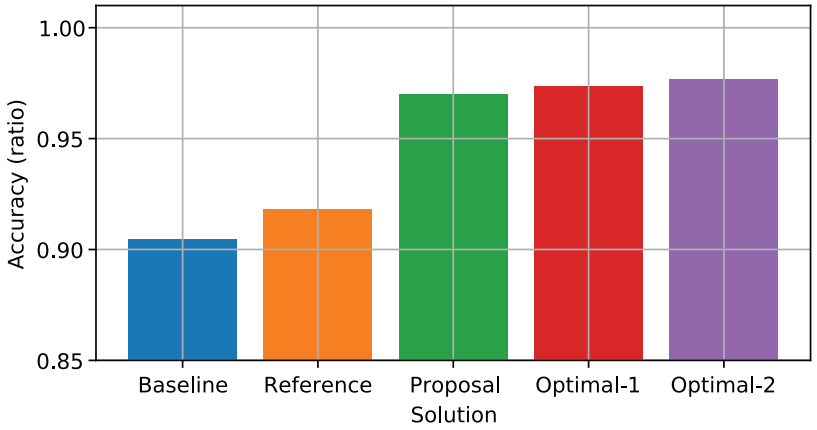

Figure 13: Enforcement accuracies obtained using the model with product composition

Table 9 summarizes the enforcement accuracies obtained through experiments, the application of the linear composition model, and the application of the product composition model. For reference, the table also includes the predictions obtained for the optimal solutions introduced in Section 3.5. Observing that our proposal reaches 99% of accuracy while the model predicted over 97% of accuracy, and applying a similar difference to the 97.72% of accuracy expected for the second optimal solution, allows us to determine that it is feasible for a future solution to reach 99.99% of accuracy, which is highly desirable for the network ecosystem.

Table 9: Enforcement accuracies

Method	Baseline	Reference	Proposal	Optimal-1	Optimal-2
Experimental	0.8976	0.9002	0.9950		
Linear Composition	0.9048	0.9182	0.9704	0.9740	0.9772
Product Composition	0.9047	0.9180	0.9700	0.9734	0.9766

6 Conclusion

In the present study, we formulated a model to predict the accuracy ratio of a system for enforcing multi-domain network intent policies using the descriptor values obtained from its design. We adjusted the coefficients of the model using experimentation with related work. Then, using the model with a goal solver mechanism, we obtained sets of values that could be used to construct a system with increased enforcement accuracy.

However, implementing the values was not trivial. We had to apply well-known patterns, practices, and theories on system operation and management to our design. We gathered and presented them as a set of design principles that should be followed to construct consistent management systems. A key aspect of these principles is that they favor the application of category theory in the conceptual definition, design, and implementation of the system

Through the application of the theorems formulated in this study, we had a consistent and verifiable system development path. On the one hand, we obtained a system design from an abstract conceptual view of the target functions. On the other hand, we obtained an implementation that realized all requirements, including the improvement of network intent enforcement accuracy. The resulting system was 10% more accurate than related work, and all network intents, even the most complex, were enforced in less than 1.75 s in a multi-domain platform comprising sites located on almost opposite sides of the world.

Our design process was based on the formulation of categorical relations—namely, morphisms, functors, and natural transformations—that connected basic concepts. For instance, we defined morphisms between the concepts present in the input network intents and the concepts present in the policy sets related to them. These categorical relations were composed to construct more complex relations. These, in turn, were composed to construct even more complex relations. The process continued until the policy enforcement concepts and relations among them were fully captured. This way, we ensured the verifiability of the resulting system.

Particularly, the correction of policy enforcement operations was achieved through the instantiation of multiple interconnected CCLs. Such interconnection was mediated by VPs. There was no hierarchy, but side-by-side collaboration among the components that implemented the CCLs, forming a graph. The potential complexity of the system state that a graph of CCLs could result in was tackled by the use of stateless functions. We implemented the components to exchange objects based on the state monad and do not store any state internally. This incorporated the CCL functions to the set of categorical relations of the system.

To support the applicability of our solution, in future studies, we will analyze the potential drawbacks stated in Section 4.7 and propose alternatives to ensure they do not apply. We will also adapt and/or extend our model to cover the network intent policy verification and the network service deployment systems. This will allow us to explore the application of our system in new scenarios. Moreover, we will study the implications of the application of similar strategies in the design of such systems.

Finally, it is worth noting that the nature of the automated interpretation and enforcement of network intents makes this process generally inexact, so 100% accuracy is not easily achieved. This justified the present study, and still justifies future studies, on network intent resolution and enforcement accuracy. Thus, we will work on further improving the model and system to be even closer to 100% of accuracy. In this respect, we will integrate the three models to evaluate an overall IBN system. We will fine-tune the model of this study if needed.

Acknowledgement: The authors express their sincere appreciation to all contributors of this paper. Additionally, we thank the funding entities whose support made this research possible.

Funding Statement: This work has been supported by the Spanish Ministry of Science and Innovation under the project ONOFRE-4, Grant PID2023-148104OB-C43, funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU.; by Horizon project RIGOUROUS funded by the European Commission, GA: 101095933; and by the Spanish Ministry of Science and Innovation under the DIN2019-010827 Industrial PhD Grant, co-funded by Odin Solutions S.L.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Ana Hermosilla, Pedro Martinez-Julia, Diego R. Lopez, Antonio F. Skarmeta; software: Ana Hermosilla, Pedro Martinez-Julia; data collection: Pedro Martinez-Julia; analysis and interpretation of results: Ana Hermosilla, Pedro Martinez-Julia; draft manuscript preparation: Ana Hermosilla, Pedro Martinez-Julia; final manuscript preparation: Ana Hermosilla, Pedro Martinez-Julia, Diego R. Lopez, Antonio F. Skarmeta; funding acquisition: Antonio F. Skarmeta. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study would be available from the Corresponding Author, A.H., upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Li C, Havel O, Olariu A, Martinez-Julia P, Nobre J, Lopez D. Intent Classification [Internet]. 2022 [cited 2025 Oct 28]. Available from: https://rfc-editor.org/rfc/rfc9316.txt.
2. OSM. OSM release five technical overview. In: OSM white paper. Sophia Antipolis, France: ETSI; 2019.
3. Adaptive Analytics I. Object constraint language [Internet]. 2024 [cited 2025 Oct 28]. Available from: https://www.omg.org/spec/OCL/2.4.
4. Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M. SWRL: a semantic web rule language combining OWL and RuleML [Internet]. 2004 [cited 2025 Oct 28]. Available from: https://www.w3.org/submissions/SWRL/.
5. Terrasse MN, Savonnet M. Formalization of the UML metamodel: an approach based upon the four-layer metamodeling architecture. In: ECOOP '00: proceedings of the workshops, panels, and posters on object-oriented technology. Berlin/Heidelberg, Germany: Springer-Verlag; 2000.
6. Klyne G, Carroll JJ. Resource description framework (RDF): concepts and abstract syntax [Internet]. 2022 [cited 2025 Oct 28]. Available from: http://www.w3.org/TR/rdf-concepts/.
7. Bjorklund M. YANG-a data modeling language for the network configuration protocol (NETCONF) [Internet]. 2010 [cited 2025 Oct 28]. Available from: https://rfc-editor.org/rfc/rfc6020.txt.
8. Liu W, Xie C, Strassner JC, Karagiannis G, Klyus M, Bi J, et al. Policy-based management framework for the simplified use of policy abstractions (SUPA) [Internet]. 2018 [cited 2025 Oct 28]. Available from: https://rfc-editor.org/rfc/rfc8328.txt.
9. de Sousa NFS, Rothenberg CE. CLARA: closed loop-based zero-touch network management framework. In: Proceedings of the 2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN); 2021 Nov 9–11; Heraklion, Greece. p. 110–5.
10. ZSM E. Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 1: Enablers [Internet]. 2021 [cited 2025 Oct 28]. Available from: https://portal.etsi.org/webapp/workprogram/Report_WorkItem.asp?WKI_ID=58053.
11. Zhang J, Yang C, Dong R, Wang Y, Anpalagan A, Ni Q, et al. Intent-driven closed-loop control and management framework for 6G open RAN. IEEE Internet Things J. 2024;11(4):6314–27. doi:10.1109/jiot.2023.3312795.
12. Baktir AC, Junior ADN, Zahemszky A, Likhyani A, Temesgene DA, Roeland D, et al. Intent-based cognitive closed-loop management with built-in conflict handling. In: Proceedings of the 2022 IEEE 8th International Conference on Network Softwarization (NetSoft); 2022 Jun 27–Jul 1; Milan, Italy. p. 73–8.

13. Craven R, Lobo J, Lupu E, Russo A, Sloman M. Policy refinement: decomposition and operationalization for dynamic domains. In: Proceedings of the 2011 7th International Conference on Network and Service Management; 2011 Oct 24–28; Paris, France. p. 1–9.
14. Abbas K, Khan TA, Afaq M, Song WC. Network slice lifecycle management for 5G mobile networks: an intent-based networking approach. IEEE Access. 2021;9:80128–46. doi:10.1109/access.2021.3084834.
15. Chowdhury M. Accelerator: an intent-based intelligent resource-slicing scheme for SFC-based 6G application execution over SDN- and NFV-empowered zero-touch network. Front Commun Netw. 2024;5:1385656. doi:10.3389/frcmn.2024.1385656.
16. Rajab ME, Yang L, Shami A. Zero-touch networks: towards next-generation network automation. Comput Netw. 2024;243(2):110294. doi:10.1016/j.comnet.2024.110294.
17. Bonfim M, Freitas F, Fernandes S. A semantic model to assist policy refinement mechanisms for NFV-MANO systems. In: Anais do VIII Workshop Pré-IETF/IRTF. Porto Alegre, RS, Brasil: SBC; 2021. p. 27–40 doi:10.5753/wpietf.2021.15780.
18. Motik B, Patel-Schneider PF, Persia B. OWL 2 web ontology language-structural specification and functional-style syntax (Second Edition) [Internet]. 2012 [cited 2025 Oct 28]. Available from: http://www.w3.org/TR/owl2-syntax/.
19. Mercian A, Kiran M, Pouyoul E, Tierney B, Monga I. INDIRA: 'Application Intent' network assistant to configure SDN-based high performance scientific networks. In: Proceedings of the 2017 Optical Fiber Communications Conference and Exhibition (OFC); 2017 Mar 19–23; Los Angeles, CA, USA. p. 1–3.
20. Cinmere I, Mehmood K, Kralevska K, Mahmoodi T. Direct-conflict resolution in intent-driven autonomous networks. arXiv:2401.08341. 2024.
21. Muktadir AHA, Jibiki M, Martinez-Julia P, Kafle VP. Repeated leader follower game for managing cloud networks with limited resources. IEEE Access. 2019;7:108174–88. doi:10.1109/access.2019.2933031.
22. Massa J, Forti S, Paganelli F, Dazzi P, Brogi A. Declarative provisioning of virtual network function chains in intent-based networks. In: Proceedings of the 2023 IEEE 9th International Conference on Network Softwarization (NetSoft); 2023 Jun 19–23; Madrid, Spain. p. 522–7.
23. Borsatti D, Cerroni W, Clayman S. From category theory to functional programming: a formal representation of intent. In: Proceedings of the 2022 IEEE 8th International Conference on Network Softwarization (NetSoft); 2022 Jun 27–Jul 1; Milan, Italy. p. 31–6.
24. de Trizio F, Sciddurlo G, Cianci I, Piro G, Boggia G. Optimizing key value indicators in Intent-Based Networks through digital twins aided service orchestration mechanisms. Comput Commun. 2024;228(2):107977. doi:10.1016/j.comcom.2024.107977.
25. Sharma Y, Bhamare D, Sastry N, Javadi B, Buyya R. SLA management in intent-driven service management systems: a taxonomy and future directions. ACM Comput Surv. 2023 Jul;55(13s):1–38. doi:10.1145/3589339.
26. Zhu Y, Chen D, Zhou C, Lu L, Duan X. A knowledge graph based construction method for Digital Twin Network. In: Proceedings of the 2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI); 2021 Jul 15–Aug 15; Beijing, China. p. 362–5.
27. Martinez-Julia P, Kafle VP, Asaeda H. Application of category theory to network service fault detection. IEEE Open J Commun Soc. 2024;5(1):4417–43. doi:10.1109/ojcoms.2024.3425831.
28. Kerboeuf S, Porambage P, Jain A, Rugeland P, Wikström G, Ericson M, et al. Design methodology for 6G end-to-end system: Hexa-X-II perspective. IEEE Open J Commun Soc. 2024;5(6):3368–94. doi:10.1109/ojcoms.2024.3398504.
29. Kostopoulos A, Christofis L, Chochliouros I, Cosmas J, De Capitani di Vimercati S, Giannoulis I, et al. Secure, privacy-preserving, and trustworthy networks. In: Towards sustainable and trustworthy 6G: challenges, enablers, and architectural design. Boston, MA, USA: Now Publishers; 2023. p. 322–56 doi:10.1561/9781638282396.
30. Martinez-Julia P, Jeong J. Intent translation engine [Internet]. 2024 [cited 2025 Oct 28]. Available from: https://datatracker.ietf.org/doc/draft-pedro-ite/.
31. Fong B, Spivak DI. Seven sketches in compositionality: an invitation to applied category theory. arXiv:1803.05316. 2018.
32. Milewski B. Category theory for programmers. San Francisco, CA, USA: Blurb; 2018.

33. Galis A, Clayman S, Mamatas L, Loyola JR, Manzalini A, Kuklinski S, et al. Softwarization of future networks and services-programmable enabled networks as next generation software defined networks. In: Proceedings of the IEEE SDN for Future Networks and Services (SDN4FNS). Washington, DC, USA: IEEE; 2013. p. 1–7.
34. Kephart JO, Chess DM. The vision of autonomic computing. Computer. 2003;36(1):41–50. doi:10.1109/mc.2003.1160055.
35. Leivadeas A, Falkner M. A survey on intent-based networking. IEEE Commun Surv Tutor. 2023;25(1):625–55.
36. Camarillo-Villa JD, Limon X, Cortes-Verdin K, Sanchez-García AJ. Functional programming oriented software design: a systematic literature review. In: Proceedings of the 2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT); 2023 Nov 6–10; León, Mexico. p. 35–44.
37. Gram. Deal [Internet]. 2023 [cited 2025 Oct 28]. Available from: https://deal.readthedocs.io/index.html.
38. Korner P, Leuschel M, Barbosa J, Costa VS, Dahl V, Hermenegildo MV, et al. Fifty years of prolog and beyond. Theory Pract Log Program. 2022;22(6):776–858. doi:10.1017/s1471068422000102.
39. Swift T, Andersen CF. The janus system: multi-paradigm programming in prolog and python. arXiv:2308.15893. 2023.
40. Niknejad N, Ismail WB, Ghani I, Nazari B, Bahari M, Hussin ARC. Understanding service-oriented architecture (SOA): a systematic literature review and directions for further investigation. Inf Syst. 2020;91(3):101491. doi:10.1016/j.is.2020.101491.
41. Movahedi Z, Kianpisheh S, Badonnel R. A survey on self-adaptation and self-healing for autonomous networks. IEEE Commun Surv Tutor. 2021;23(3):1802–37.
42. Gil Herrera J, Botero JF. Resource allocation in NFV: a comprehensive survey. IEEE Trans Netw Serv Manage. 2016;13(3):518–32. doi:10.1109/tnsm.2016.2598420.
43. Fabregat-Hernández A, Palanca J, Botti VJ. Exploring explainable AI: category theory insights into machine learning algorithms. Mach Learn Sci Technol. 2023;4(4):045061. doi:10.1088/2632-2153/ad1534.
44. Rutten E, Marchand N, Simon D. Feedback control as MAPE-K loop in autonomic computing. In: de Lemos R, Garlan D, Ghezzi C, Giese H, editors. Software engineering for self-adaptive systems III. Assurances. Cham, Switzerland: Springer International Publishing; 2017. p. 349–73. doi:10.1007/978-3-319-74183-3_12.
45. Martinez-Julia P, Kafle VP, Asaeda H. Intelligent network service automation: improving accuracy and efficiency in network management. IEEE Trans Netw Serv Manage. 2025;22(4):2987–3002. doi:10.1109/tnsm.2025.3541225.
46. The Hy Language Authors. Hy Language [Internet]. 2024 [cited 2025 Oct 28]. Available from: http://hylang.org/hy/doc/v1.0.0.
47. The SWI Prolog Authors. SWI Prolog [Internet]. 2025 [cited 2025 Oct 28]. Available from: https://www.swi-prolog.org/.
48. Beckett D, Berners-Lee T, Prud'hommeaux E, Carothers G. RDF 1.1 Turtle—Terse RDF triple language [Internet]. 2014 [cited 2025 Oct 28]. Available from: http://www.w3.org/TR/turtle/.
49. Huawei-IOAM. YANG2RDF [Internet]. 2025 [cited 2025 Oct 28]. Available from: https://github.com/Huawei-IOAM/yang2rdf.
50. Martinez-Julia P, Kafle VP, Asaeda H. iCPN: scalable control plane for the network service automation system. In: Proceedings of the 2024 IFIP/IEEE Network Operations and Management Symposium (NOMS). Washington, DC, USA: IEEE; 2024. p. 1–5.