Journal of the Brazilian Computer Society, 2025, 31:1, doi: 10.5753/jbcs.2025.5453

L-PRISM: A Domain-Specific Language for Describing Multimedia Service Function Chains

Franklin J. V. Quico [**Fluminense Federal University** | ***fventuraq@midiacom.uff.br***]
Anselmo L. E. Battisti [**Fluminense Federal University** | ***anselmo@midiacom.uff.br***]
Débora Muchaluat-Saade [**Fluminense Federal University** | ***debora@midiacom.uff.br***]
Flavia C. Delicato [**Fluminense Federal University**| ***fdelicato@ic.uff.br***]

MídiaCom Lab, Institute of Computing, Universidade Federal Fluminense, Av. Gal. Milton Tavares de Souza, s/n, São Domingos, Niterói, RJ, 24210-590, Brazil.

Received: 19 January 2025 • **Accepted:** 19 May 2025 • **Published:** 05 August 2025

Abstract Virtualization has emerged as a key technology for handling the complexity of heterogeneous environments, such as the Internet of Things (IoT) and multimedia systems. In this context, multimedia sensors represent an important data source that contributes to the development of the Internet of Media Things (IoMT) paradigm. Using the concepts of virtualization and IoMT, a multimedia Virtual Network Function (*multimedia VNF*) has been introduced to encapsulate virtualized devices and software components for processing multimedia streams. Frequently, multimedia streams require a sequence of processing steps, forming what is known as multimedia Service Function Chains (*multimedia SFCs*). With the advancement of technologies like augmented reality and autonomous vehicles, which require processing complex multimedia streams, often handled by *multimedia SFCs*, the deployment continues to be highly challenging. Currently, the execution of *multimedia SFCs* is performed by creating and binding components individually, requiring manual configuration and low-level integration. This increases the development time and probability of errors. Moreover, existing approaches lack a standardized mechanism for describing *multimedia SFCs*, resulting in ad hoc solutions with low interoperability. Despite the importance of describing *multimedia SFCs*, there has been limited research on this topic. To bridge this gap, we propose a novel metamodel named M-PRISM, designed to serve as the conceptual foundation for describing multimedia stream processing. Using this metamodel, we create a new Domain-Specific Language (DSL) named L-PRISM, tailored to describe *multimedia SFCs*. Additionally, we introduce a novel architecture for executing *multimedia SFCs*, demonstrated through a Proof-of-Concept (PoC) which follows the proposed architecture and executes *multimedia SFCs* described in L-PRISM. Our approach was evaluated through experiments with software developers, focusing on aspects such as expressiveness, ease of adoption, and reduction of configuration complexity. In the experiments we adopted the Goal Question Metric (GQM), Technology Acceptance Model (TAM) and Cognitive Dimensions of Notations (CDN) approaches, and the results indicate that L-PRISM and its PoC facilitate the definition and deployment of *multimedia SFCs* based on *multimedia VNFs*.

Keywords: IoMT, IoT, VNF, SFC, DSL, M-PRISM, L-PRISM

1 Introduction

Virtualization is a concept that has paved the way for numerous research studies in Cloud Computing, 5G, and IoT [Zerifi *et al.*, 2025; Yi *et al.*, 2018]. This technology not only reduces the costs of implementing and managing infrastructures, but also serves as a fundamental pillar in paradigms such as Network Function Virtualization (NFV) [Mijumbi *et al.*, 2016; Santos *et al.*, 2022a]. The adoption of NFV and its Virtual Network Function (VNF) has been shown to be effective in reducing the consumption of computational and network resources. Moreover, virtualization increases the speed and efficiency of resource provisioning in cloud and edge environments, facilitating resource management and orchestration [Yi *et al.*, 2018].

Many researches combine NFV technology with multimedia stream processing. In [Battisti *et al.*, 2021], the authors explore the concept of virtual devices that abstract the heterogeneity of multimedia sensors and introduce Virtual Multimedia Sensors (VMS), which we will refer to as *multimedia VNFs* in our work. These *multimedia VNFs* enable multimedia stream processing using lightweight virtualization, a method that provides process isolation while sharing the same OS kernel, resulting in better performance, faster initialization, and lower resource consumption compared to traditional virtual machines [Sisinni *et al.*, 2024]. Furthermore, the authors demonstrated the feasibility of using chained *multimedia VNFs* to create complex multimedia stream processing pipelines. However, this characteristic was not fully explored in their work.

In [Ghorab *et al.*, 2020], the authors investigated the joint load balancing and auto-scaling of *multimedia VNFs* in edge or cloud environments. More recently, container-based virtualization, a key lightweight virtualization approach, has been actively explored to enhance resource management in edge computing, improving latency and reducing overhead compared to traditional virtualization methods [Roges and Ferreto, 2025]. Additionally, the application of containerization techniques has expanded to IoT and multimedia services, as demonstrated in the virtualization of LoRaWAN infrastructure components, where containerized solutions improve flexibility and performance in diverse environments [Sisinni *et al.*, 2024]. In this context, virtualization concepts like NFV have been actively integrated into different studies to process multimedia streams.

The Service Function Chain (SFC) is a sequence of sorted VNFs associated with a Service Level Agreement (SLA) [Bhamare *et al.*, 2016]. By chaining multimedia functions, it is possible to create complex multimedia stream processing pipelines. When VNFs in an SFC are designed for multimedia processing, they are referred to as *multimedia SFCs*.

Multimedia SFCs have significant potential in various fields, such as IP Multimedia Subsystem (IMS), cloud gaming, real-time video analytics, and interactive streaming applications [Di Mauro *et al.*, 2021a,b]. However, they face numerous challenges, including development complexities, management difficulties, safety concerns, and limited observability. The components of a *multimedia SFC* are often created by different developers using a wide range of technologies [Mao *et al.*, 2017]. Consequently, building a *multimedia SFC* requires users to have substantial experience in these technologies. Furthermore, the interoperability of these components is typically constrained, further complicating their integration and (re)use.

A promising approach to addressing these challenges is to create a layer that abstracts the technical complexities of handling *multimedia SFCs*. Multiple alternatives can be used to achieve this goal, including frameworks [Garcia et *al.*, 2017], code generation tools [Neto *et al.*, 2017], and domain-specific languages (DSL) [Negm *et al.*, 2019]. Among these options, DSLs stand out for their ability to simplify the description of complex applications within a specific domain.

DSLs are specialized programming languages designed to efficiently express solutions and operations within a specific domain [Borum and Seidl, 2022]. DSLs allow users to express solutions more efficiently, thus reducing the time required to implement domain-specific functionalities compared to general-purpose languages. By focusing on domain concepts and hiding unnecessary technical details, DSLs make the application code easier to read, maintain, and extend. DSLs also enable the automation of repetitive tasks specific to a domain.

A DSL is defined over a metamodel, which establishes the structural and semantic rules of a domain, ensuring consistency and enabling automation. Metamodels such as TOSCA-NFV [TOSCA, 2017] and frameworks like ETSI GS NFV-SOL [ETSI, 2022] have emerged to facilitate the modeling and orchestration of NFV services. In the context of NFV, DSLs can play a crucial role in defining and managing complex service deployments. However, despite these advances, there is still a significant gap in addressing *multimedia SFCs*.

The need to focus on *multimedia SFCs* separately arises from their unique nature and challenges. Some of these challenges are (i) managing large-volume and high-speed data flows, (ii) synchronizing multiple formats and media types, and (iii) efficient real-time encoding and decoding [Nauman *et al.*, 2020]. Furthermore, *multimedia SFCs* often requires greater flexibility in terms of scalability and adaptation to dynamic changes in resource demand.

In this work, our aim is to address these gaps by proposing a metamodel called *Metamodel for Programming IoT Sensors for Multimedia (M-PRISM)* and a DSL named *Language for Programming IoT Sensors for Multimedia (L-PRISM)*. The metamodel and the language serve as the conceptual foundation for describing and creating *multimedia SFCs*, particularly in the emerging field of the IoMT. The decision to develop M-PRISM, rather than extend existing solutions such as TOSCA-NFV, was made because i) those models do not natively address the dynamic and high-performance requirements of *multimedia SFCs*, (ii) they lack efficient orchestration mechanisms, and (iii) they do not support media-type-specific resource optimization. Extending those models would result in partial, rigid, and unnecessarily complex solutions. Instead, we propose M-PRISM, a metamodel designed from the ground up to provide flexibility, expressiveness, and optimization for virtualized multimedia environments.

Our DSL leverages the M-PRISM model, which provides detailed structures for specifying the properties of *multimedia SFCs*. This model allows the specification of the types, formats, and resolutions of data streams that *multimedia VNFs* can process and transmit. Moreover, it also considers scenarios where a *multimedia VNF* can handle streams in various formats and qualities. This aspect is particularly relevant because the computational and network requirements associated with a *multimedia VNF* are variable and depend on the characteristics of the multimedia stream to be processed [Kalan and Dulger, 2024].

The characteristics of the *multimedia SFCs*, such as resources required and quality of service are essential for developers of *multimedia SFCs*, as they provide the necessary details to understand and use *multimedia VNFs*. Consequently, this will enable developers to specify their *multimedia SFCs* efficiently. Furthermore, at the application level, M-PRISM can facilitate the effective orchestration and management of *multimedia SFCs*. In addition, using a dedicated DSL offers benefits such as customization, easier implementation, and simplified maintenance of components. The main contributions of this work are:

- We propose a novel metamodel, named M-PRISM, for designing languages tailored to describe multimedia applications. The metamodel provides an abstract framework that defines the elements, properties, and relationships required to specify multimedia applications;
- Using the proposed metamodel, we define a new DSL, named L-PRISM, tailored to describe *multimedia SFCs* based on *multimedia VNFs*. The L-PRISM language leverages the structured foundation of the metamodel to enable the modeling of multimedia

workflows, facilitating the orchestration of *multimedia SFCs*;

- We create a Proof-of-Concept (PoC), named ALFA 2.0, that executes *multimedia SFCs*, described using L-PRISM, in an edge-cloud environment. Our PoC demonstrates the feasibility of deploying such *multimedia SFCs* across distributed infrastructures.

This paper is an extended and revised version of our previously published work [Quico *et al.*, 2024]. In this work, we address two additional topics. First, we present a comprehensive description of the M-PRISM metamodel that establishes the foundation of L-PRISM. Second, we present a software architecture and provide a detailed description of its components, which enable the execution of *multimedia SFCs* defined using L-PRISM. Third, we provide new quantitative and qualitative analyses of L-PRISM.

The remainder of this paper is organized as follows. Section 2 describes the background and related work. Section 3 introduces the M-PRISM metamodel, which is the basis of our proposal. Section 4 details the proposed DSL, named L-PRISM. Section 5 describes our PoC. In Section 6 and Section 7, we present both a quantitative and qualitative evaluation of our work. Section 8 presents our main conclusions and future work.

2 Background and Related Work

Traditional sensors, such as thermostats, presence detectors, and humidity detectors, typically generate discrete data in a readable format. In contrast, multimedia sensors produce continuous, complex, and large-volume data, which demand high processing speeds, significant storage capacity, wide bandwidth, low latency, and considerable energy consumption [Nauman *et al.*, 2020; Battisti *et al.*, 2020a]. These characteristics add complexity to their operation compared to traditional sensors.

The Internet of Multimedia Things (IoMT)[1] are IoT environments where the sensors are cameras and microphones with limited processing and storage capacity, which connect to heterogeneous devices and diverse applications. These environments has demonstrated substantial potential to facilitate the collection, processing, and transmission of multimedia streams using multimedia sensors [Allouche *et al.*, 2021]. Research has indicated that multimedia sensors require distinct characteristics and environments for optimal operation [Das *et al.*, 2014]. This integration highlights the need for customized approaches for devices, sensors, and systems that specifically handle multimedia streams.

A real-time multimedia application typically consists of physical devices that generate multimedia streams and software applications that directly consume these streams. This tight integration between hardware and software creates coupled systems [Khan *et al.*, 2022; Coêlho *et al.*, 2024] hindering the interoperability and scalability of the applications. To address these challenges, an effective strategy is to decouple applications from physical sensors by adopting virtual devices and Multimedia Virtual Network Functions (*multimedia VNFs*) [Battisti *et al.*, 2021].

A virtual device (VD) is a software entity that replicates the functionality of a physical device [Alam *et al.*, 2021]. VDs are capable of receiving data from physical devices and, in some cases, can also send commands back to them [Das *et al.*, 2014]. In this study, we focus specifically on VDs that receive multimedia streams from physical multimedia devices.

Multimedia VNFs are software components designed to process multimedia streams [Battisti *et al.*, 2021]. These components typically address a specific task, such as converting a video stream to multiple formats or detecting particular features in an audio stream, such as gunfire or hate speech. The combination of these two concepts can be used to create complex multimedia systems that address the presented problems. Figure 1 provides an overview of a three-tier architecture that integrates virtual devices and *multimedia VNFs* to build complex applications within a multi-node edge-cloud environment.

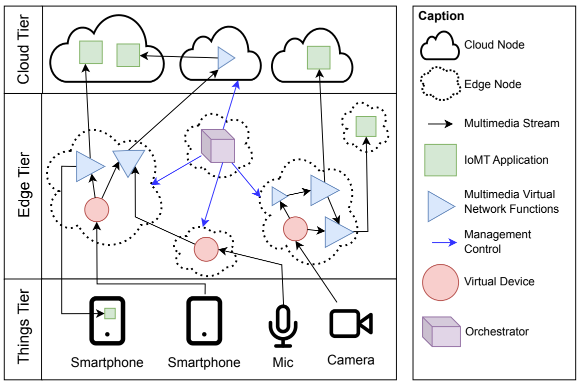


Figure 1. Three-tier architecture overview [Battisti *et al.*, 2021].

Deploying complex multimedia applications, also known as *multimedia SFCs*, involves configuring multiple components in a specific sequence [Battisti *et al.*, 2021]. This process requires developers to have in-depth knowledge of the software and hardware used to capture, process, and deliver multimedia streams to users. Describing these applications through a standard language for automated deployment can significantly reduce complexity for developers.

Several studies have explored the use of NFV and SFCs for executing multimedia applications [Farahani *et al.*, 2023; Santos *et al.*, 2022b]. Despite leveraging SFCs to achieve their objectives, these studies do not focus on the description language process. As a result, reproducing SFCs is not a trivial task.

Several description languages, metamodels, and DSLs have been developed to facilitate the definition and deployment of complex applications. These tools support the development of systems and architectures in various fields, such as IoMT and multimedia systems. DSLs, in particular, are language specifications that describe the features, syntax, and behavior of systems within a specific domain [Borum and Seidl, 2022]. While some metamodels,

[1] https://www.iso.org/news/ref2449.html

such as TOSCA-NFV [TOSCA, 2017], facilitate the definition and partial deployment of NFV services, there has been limited effort in creating metamodels or DSLs designed explicitly for the deployment and orchestration of *multimedia SFCs*.

A related work is the YANG data modeling language [Schönwälder *et al.*, 2010], designed to describe network configurations and telecommunication services. YANG supports NETCONF-based operations, including configuration, state data management, remote procedure calls (RPCs), and notifications. YANG models can be translated into XML syntax, enabling applications using XML parsers to operate and manipulate these models [Björklund, 2010]. However, YANG lacks the features necessary to describe *multimedia SFCs*, limiting its applicability in this domain.

Another related work is the TOSCA-NFV metamodel [TOSCA, 2017]. This is a specific data model designed to work with virtualization technologies, enabling the description, deployment, and management of NFV-based applications and services. TOSCA-NFV details the components of its solutions, their interactions, and dependencies. It uses a topology template to define workload components and a relationship template to specify their interconnections. However, TOSCA-NFV primarily focuses on traditional virtualization systems and does not adapt well to lightweight virtualization platforms such as Docker and Kubernetes.

Building on TOSCA-NFV, Mininet-NFV [Castillo-Lema *et al.*, 2019] introduces an advanced framework for NFV orchestration, facilitating the implementation and operation of network services based on VNFs. Mininet-NFV extends the capabilities of Mininet—a widely recognized tool for agile experimentation with networks, SDN, and NFV—by supporting parameterized TOSCA-NFV templates and virtual link descriptors, enabling detailed and flexible network configurations. However, like TOSCA-NFV, it lacks the features necessary to describe and execute *multimedia SFCs*.

Similarly, ETSI GS NFV-SOL 001 [ETSI, 2022] builds on TOSCA-NFV to meet the specification requirements defined in ETSI GS NFV IFA 011 [011, 2023] and ETSI GS NFV IFA 014 [014, 2021]. Those standards define descriptors for Virtualized Network Functions Descriptor (VNFDs), Network Services (NSDs), and Physical Network Functions (PNFDs). Like TOSCA-NFV, ETSI GS NFV-SOL 001 specifies requirements for the management and orchestration of VNFs, maintaining a focus on traditional virtualization approaches.

Although very related to our proposal, neither TOSCA-NFV nor ETSI NFV-SOL specifically addresses the requirements of multimedia applications and systems, particularly concerning *multimedia SFCs* and their components. Therefore, while our work builds upon the principles of TOSCA-NFV, it distinguishes itself by addressing the critical aspects required for the definition, orchestration, and management of *multimedia SFCs* implemented on distributed platforms, leveraging edge or cloud computing with support for lightweight virtualization.

Table 1 compares related languages, templates, and data models that support the definition of SFCs. The last row of the table presents our proposed DSL named L-PRISM. The meaning of each column in the table is as follows:

- **IoMT:** analyzes whether these works are specifically tailored for IoMT applications, with a particular focus on enabling the description of *multimedia SFCs* based on *multimedia VNFs*;
- **Edge Computing:** identifies if these works are tailored for edge computing environments;
- **Light Virtualization:** evaluates the support for lightweight virtualization platforms such as Docker and Kubernetes.

Table 1. Comparison of languages used to describe SFCs.

Related Work	**IoMT**	**Edge Computing**	**Light Virtualization**
YANG [Schönwälder *et al.*, 2010]	-	-	-
TOSCA-NFV [TOSCA, 2017]	-	✓	-
Mininet-NFV [Castillo-Lema *et al.*, 2019]	-	✓	✓
ETSI NFV-SOL [ETSI, 2022]	-	✓	✓
L-PRISM	✓	✓	✓

In our previous work [Battisti *et al.*, 2020c, 2021], we proposed an architecture that enables the execution and management of *multimedia VNFs* based on lightweight virtualization in an edge-cloud environment. Additionally, we presented a PoC called ALFA, capable of running *multimedia VNFs* in a multi-node edge-cloud environment. However, at that time, we did not explore *multimedia SFCs* nor provide a metamodel or DSL to describe them, which would facilitate their creation and maintenance. In this work, we address this limitation by simplifying the definition of *multimedia SFCs* through the development of a metamodel and a DSL.

3 M-PRISM

This section presents M-PRISM, a metamodel designed to serve as the conceptual foundation for describing multimedia applications. This metamodel draws inspiration from established approaches, including [TOSCA, 2017], while addressing the specific requirements of multimedia applications described in Section 2.

Figure 2 presents M-PRISM as a class diagram which outlines the attributes and relationships necessary to model a multimedia application. The diagram elements are organized and distinguished by color, as follows:

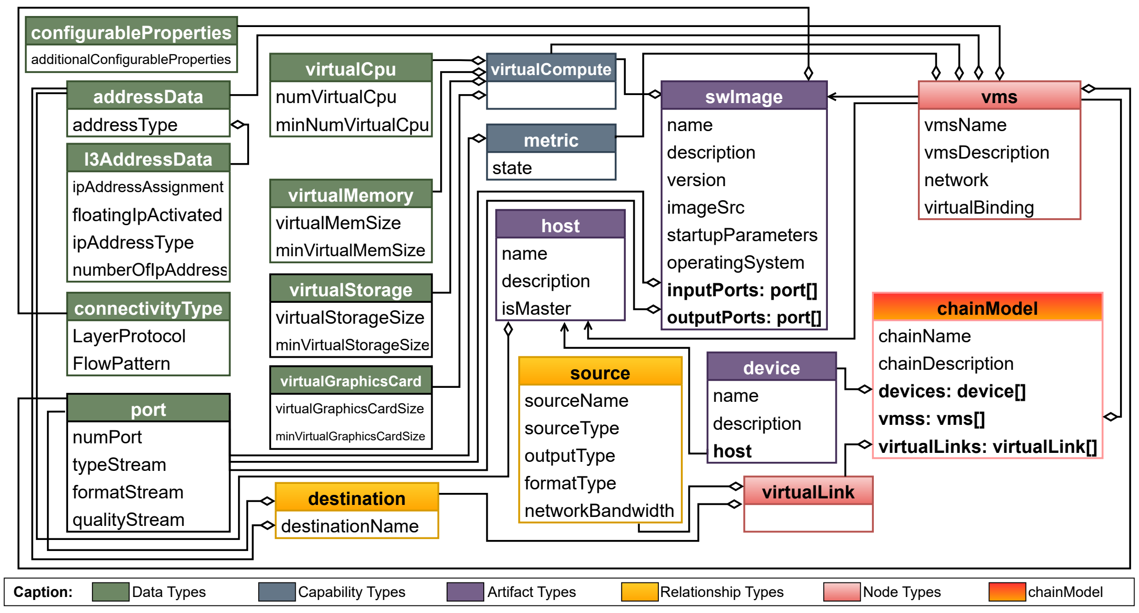


Figure 2. Class diagram of the M-PRISM metamodel.

- **Elements in green:** Represent the data types detailed in Section 3.1. These data types serve as the foundation for more complex structures within M-PRISM, modeled through aggregation functions;
- **Elements in gray:** Represent the capabilities that the components of a multimedia application can have, as described in Section 3.2. For example, the *virtualCompute* element groups the computational capabilities associated with a virtualized component;
- **Elements in purple:** Represent artifacts that can be used within a multimedia application, as discussed in Section 3.3. In distributed environments, components can be deployed on different servers within a network (hosts). Therefore, when describing a *multimedia VNF* within a *multimedia SFC*, it is necessary to specify the node where it will be implemented;
- **Elements in yellow:** Represent the communication components that enable interaction between the different elements of a *multimedia SFC* (VDs and *multimedia VNFs*), as presented in Section 3.4. For example, the *source* element defines the attributes needed to configure the source of a multimedia stream, while the *destination* element specifies the configuration for the destination of the stream;
- **Elements in pink:** Represent the components created within a *multimedia SFC* as will be detailed in Section 3.5. For our domain, we have defined two main components:
  - *vms*: Represents a *multimedia VNF* as a lightweight virtualized component, commonly implemented using containers or other similar lightweight virtualization technologies;
  - *virtualLink*: Represents the network configuration between two virtualized components.
- **Element in orange:** Represents the main element of our language, called *chainModel*. This element comprehensively describes a *multimedia SFC* by grouping all the aforementioned components, including virtual devices, virtual links, and *multimedia VNFs*, as commented in Section 3.6.

3.1 Data Types

The components that constitute the data types primarily serve to standardize and manage essential aspects of network configuration, resource allocation, and multimedia stream handling within virtualized environments. These components have been designed with reference to TOSCA-NFV, ETSI-NFV, and multimedia systems operations, with the goal of enhancing the flexibility, scalability, and efficiency of *multimedia VNFs*. This will enable detailed control over the allocation of IP addresses, CPU, memory, storage, and connectivity parameters. Within the M-PRISM context, these data types extend and refine conventional configurations, addressing the unique requirements of multimedia applications, such as the need for high-quality graphics, variable stream formats, and customizable resources. In the next section, the attribute names marked with an asterisk (*) are mandatory.

3.1.1 l3AddressData

This data type is primarily used to configure IP addresses within a subnet, specifying whether they should be manually assigned or automatically managed by the orchestrator. This functionality ensures that address allocation complies with architectural constraints. Although the full *l3AddressData* structure is supported in M-PRISM, our initial implementation focuses on key attributes due to testing environment limitations. Table 2 summarizes the attributes of *l3AddressData*.

Table 2. Attributes of *l3AddressData*.

Name	Description
ipAddress-Assignment* **Type:** *Boolean*	Specifies whether IP address assignment is manual or automatic.
floatingIp-Activated **Type:** *Boolean*	Determines if the floating IP functionality is enabled.
ipAddressType* **Type:** *String*	Indicates the type of IP address, allowing values such as ipv4 and ipv6.
numberOf-IpAddress **Type:** *Integer*	Defines the minimum number of IP addresses to be assigned.

3.1.2 addressData

This data type provides information on the address assigned to the connection point and allows for flexible management of address assignment and connection types. The *addressData* structure comprises attributes such as addressType and *l3AddressData*, offering flexibility and advanced management capabilities in networks. Table 3 presents the attributes of *addressData*.

Table 3. Attributes of *addressData*.

Name	Description
addressType* **Type:** *String*	Describes the address assigned to the connection point, which can be managed by the orchestrator or assigned manually, depending on the state of the ipAddressAssignment property in *l3AddressData*.
l3AddressData **Type:** *mprism.datatype .l3AddressData*	Provides complementary information about the assigned address, enhancing network management capabilities.

3.1.3 connectivityType

This is a complex data type used to describe the type of connection between connection points, which is essential for creating multimedia services. Unlike TOSCA-NFV, which employs *connectivityType* to create virtual links (*virtualLink*), M-PRISM employs this data type to specify, within virtual machine images (*swImage*), the type of connections they can accept and transmit. This enables end users to understand the types of connections that may exist between the endpoints of elements within a *multimedia SFC*. Table 4 presents the attributes of *connectivityType*.

3.1.4 virtualCpu

This data type describes the CPU properties at each connection point. It allows for manipulation of the assigned processor properties, providing flexibility in virtual resource configuration. Table 5 presents the attributes of *virtualCpu*.

Table 4. Attributes of *connectivityType*.

Name	Description
layerProtocol* **Type:** *String*	Identifies the connection protocol used, such as ethernet, ipv4, ipv6, among others.
flowPattern **Type:** *String*	Defines the connection mode, with valid values such as Line, Tree, and Mesh.

Table 5. Attributes of *virtualCpu*.

Name	Description
numVirtualCpu **Type:** *Integer*	Property defining the number of processors assigned to the connection point (*multimedia VNF*). By default, this value is equal to minNumVirtualCpu but can be modified when creating the *multimedia SFC*.
minNum-VirtualCpu* **Type:** *Integer*	Property proposed by M-PRISM as an extension to the TOSCA-NFV structure, allowing the minimum number of CPUs required for the connection point to be defined. This property must be set when creating the virtual machine image (*swImage*).

3.1.5 virtualMemory

This data type describes the properties of the memory assigned to the virtual machine (*multimedia VNF*) and provides flexibility in memory resource configuration. Table 6 presents the attributes of *virtualMemory*.

Table 6. Attributes of *virtualMemory*.

Name	Description
virtualMemSize **Type:** *Integer*	Defines the amount of memory, in megabytes (MB), assigned to the connection point. This integer property, by default, takes the same value as minVirtualMemSize but can be modified when creating a *multimedia SFC*.
minVirtual-MemSize* **Type:** *Integer*	Property proposed by M-PRISM as an extension to the TOSCA-NFV structure, allowing the minimum amount of memory required for the connection point to be specified. This property must be set when creating the virtual machine image (*swImage*).

3.1.6 virtualStorage

This data type is defined in our metamodel. Although TOSCA-NFV does not describe a specific data type for storage, it does consider storage in virtual machine deployments. In M-PRISM, this data type has been included to meet the needs of virtualized multimedia applications, which require sufficient space to store multimedia data, application data (e.g., code and files), and associated temporary files. Table 7 presents the attributes of *virtualStorage*.

Table 7. Attributes of *virtualStorage*.

Name	Description
virtual-StorageSize **Type:** *Integer*	Defines the amount of storage, in megabytes (MB), assigned to the connection point. This integer property, by default, takes the same value as minVirtualStorageSize but can be modified when creating a *multimedia SFC*.
minVirtual-StorageSize* **Type:** *Integer*	Property that allows defining the minimum storage amount required for the connection point. This property must be set when creating the virtual machine image (*swImage*).

3.1.7 virtualGraphicsCard

This data type specifies if *multimedia VNFs* must be executed in a node that contains a GPU with certain resources. TOSCA-NFV does not have a specific data type for specifying such property. Therefore, M-PRISM enables applications to meet GPU specifications. Table 8 presents the attributes of *virtualGraphicsCard*.

Table 8. Attributes of *virtualGraphicsCard*.

Name	Description
virtualGraphics-CardSize **Type:** *Integer*	Defines the amount of graphics card memory in megabytes *(MB)* assigned to the connection point. This integer property, by default, takes the same value as minVirtualGraphicsCardSize but can be modified when creating a *multimedia SFC*.
minVirtual-Graphics-CardSize **Type:** *Integer*	Property that allows specifying the minimum amount of graphics memory (VRAM) that must be assigned to the connection point. This property must be set when creating the virtual machine image (*swImage*).

3.1.8 configurableProperties

This is a data type that describes the configurable properties of an element. In the context of M-PRISM, both virtual devices (VD) and *multimedia VNFs* can have configurable properties set before initialization. This data type is essential for adjusting the specific characteristics of virtualized elements and ensuring they meet the needs of multimedia applications. Table 9 presents the attributes of *configurableProperties*.

Table 9. Attributes of *configurableProperties*.

Name	Description
additional-Configurable-Properties **Type:** *String*	Defines a variable that can be set by the user. This property is commonly used in the software image (*swImage*) description.

3.1.9 port

This is a data type that describes an input or output port enabled in a VMS. In M-PRISM, this port has features oriented toward handling multimedia streams, making it essential for applications requiring video, audio, image, or text transmission. Besides the port number, properties have been added to define the type, format, and quality of the multimedia stream processed through the specified port. Table 10 presents the attributes of *port*.

Table 10. Attributes of *port*.

Name	Description
numPort* **Type:** *Integer*	Enabled port number. For example 10001.
typeStream **Type:** *String*	Type of multimedia stream used by this port. Possible values include video, audio, image, and text.
formatStream **Type:** *list*	List of multimedia stream formats that can be used on this port. For example: H.264, H.265, VP9 (video formats).
qualityStream **Type:** *list*	List of multimedia stream qualities that can be used on this port. For example: 720p, 1080p, 4K (video resolutions).

3.2 Capability Types

The capability types in M-PRISM are designed to enhance virtualized component management and operational efficiency within *multimedia SFCs*. These capabilities enable monitoring, resource allocation, and detailed configuration of virtualized elements, ensuring compliance with TOSCA-NFV standards while addressing the specific demands of *multimedia VNFs*. By leveraging these capabilities, M-PRISM provides precise control over monitoring activities and resource distribution, meeting the unique requirements of scalable and high-performance multimedia applications.

3.2.1 metric

This is a capability type used to monitor a virtualized element. This capability is essential when a *multimedia VNF* needs to send operational reports. The *metric* capability has been adapted to meet the specific needs of *multimedia SFCs* while maintaining compliance with the TOSCA-NFV definition. Table 11 presents the attributes of *metric*.

Table 11. Attributes of *metric*.

Name	Description
state* **Type:** *Boolean*	Indicates the monitoring status of the element, represented as a boolean value specifying whether the monitoring capability is enabled.
listPorts **Type:** *list*	A list of ports that can transmit monitoring data. Each entry in this list is of the *port* type, as defined previously.

3.2.2 virtualCompute

Represents the computational resources that can be allocated to a virtualized component. These resources can be configured directly in the *multimedia VNF* or preconfigured in the software image registry. The properties of *virtualCompute* allow for managing various hardware resources assigned to the component, such as memory, CPU, storage, and, optionally, a graphics card. Table 12 presents the attributes of *virtualCompute*.

Table 12. Attributes of *virtualCompute*.

Name	Description
virtualMemory* **Type:** *mPrism.datatype .virtualMemory*	Describes the minimum amount of RAM required by the software images and the real amount allocated to the virtualized component, as detailed in Section 3.1.5.
virtualCpu* **Type:** *mPrism.datatype .virtualCpu*	Describes the minimum number of CPUs required by the software images and the real number allocated to the virtualized component, as detailed in Section 3.1.4.
virtualStorage* **Type:** *mPrism.datatype .virtualStorage*	Describes the minimum amount of storage required by the software images and the real amount allocated to the virtualized component, as detailed in Section 3.1.6.
virtual-GraphicsCard **Type:** *mPrism.datatype .virtualGraphics Card*	Describes the minimum amount of graphical memory required (if needed) by the software image and the real amount allocated to the virtualized component, as detailed in Section 3.1.7.

3.3 Artifact Types

The artifacts in M-PRISM are essential tools designed to support developers in creating and managing virtualized environments. These tools include software images (*swImage*), hosts (*host*), and virtualized devices (*vd*). Unlike other elements in our metamodel, artifacts are shared resources available to all developers within the system. Together, these artifacts enable efficient virtualization, optimal resource management and scalability, addressing the specific needs of multimedia stream processing.

3.3.1 swImage

This is an artifact defined in our M-PRISM metamodel, primarily based on the TOSCA-NFV metamodel. This artifact represents a data model that describes a system image used to virtualize a *multimedia VNF*. Its primary purpose is to catalog software images developed by the community, providing precise and detailed information for use by other developers. However, it does not represent a specific component; *swImage* acts as a repository containing essential details for the configuration and operation of software images. Table 13 presents the attributes of *swImage*.

Table 13. Attributes of *swImage*.

Name	Description
name* **Type:** *String*	The name of the image.
description **Type:** *String*	A description of the image.
version* **Type:** *String*	The version of the image.
ImageSrc* **Type:** *String*	A reference to the location where the image is stored in the file system.
startupParameters **Type:** *mPrism .datatype .configurable Properties*	Initialization variables required by the image. The image author defines these and can be configured by the developer.
operatingSystem* **Type:** *String*	The operating system used by the software image.
virtualCompute* **Type:** *mPrism .capabilities .virtualCompute*	Describes the minimum computational properties required by the software image, as detailed in Section 3.2.2.
inputPorts **Type:** *list*	A list of ports capable of receiving multimedia streams, including the type, format, and quality of supported streams.
outputPorts **Type:** *list*	A list of ports enabled for sending data streams, useful for performance metrics.
*connectivityType** **Type:** *mPrism .datatype .connectivityType*	Describes the software image's behavior regarding the output connection type, as detailed in Section 3.1.3.

3.3.2 host

It is an artifact representing an entity designed to provide computational resources to the applications deployed within it. Although TOSCA-NFV does not define this element in detail, in M-PRISM, a *host* is classified as an artifact intended to assist developers. According to [Battisti *et al.*, 2020c, 2021], a *host* can be defined as a small-to-medium-sized computational entity that provides computing, storage, and networking resources to the components deployed within it. In the V-PRISM architecture PoC, a host is controlled by a master host, which manages the available hosts within the network. Table 14 presents the attributes of *host*.

Table 14. Attributes of *host*.

Name	Description
name* **Type:** *String*	The name of the host.
description **Type:** *String*	A description of the host.
addressIp **Type:** *mPrism .datatype .addressData*	Describes the host's IP address. This property is defined using the *addressData* data type, as detailed in Section 3.1.2.
isMaster* **Type:** *Boolean*	Indicates whether the registered node will act as the master node within the network.

3.3.3 device

The artifact *device* is a virtual representation of a physical device (camera or microphone). One advantage of virtualized devices is that a single multimedia stream captured from a camera/microphone can be reused multiple times and sent to different destinations. Table 15 presents the attributes of *device*.

Table 15. Attributes of *device*.

Name	Description
deviceName* **Type:** *String*	Unique name of a virtual device that works as an *ID* in a *virtualLink*.
deviceId **Type:** *String*	Identifies a virtualized *device*. This index helps any element of the *multimedia SFC* to subscribe to a *device*. When a *device* receives a subscription, it replicates its stream and sends it to the new subscriber. It should be noted that virtualized devices are not created in the *multimedia SFC*. They are initialized separately and can be part of different *multimedia SFCs*.

3.4 Relationship Types

Relationship types represent the elements necessary to model the relationships between the different components of a *multimedia SFC*. These relationships are defined through properties and attributes describing the components' interaction.

3.4.1 source

Defines the properties associated with the origin point of the multimedia stream, specifying where the transmission begins. Table 16 presents the attributes of *source*.

Table 16. Attributes of *source*.

Name	Description
sourceName* **Type:** *String*	Source of a multimedia stream. This name must be identical to that configured in a *deviceName* or a *vmsName*.
sourceType* **Type:** *String*	Type of the source node of the virtual link, which can be *multimedia VNF* or *device*.
outputType* **Type:** *String*	Describes the type of multimedia stream to be sent. Multimedia applications usually process one type of stream but may produce a different output type.
formatType **Type:** *String*	The multimedia stream type such as MP4, AVI, MPEG.
network-Bandwidth **Type:** *Integer*	Bandwidth assigned to the connection. Within a host, it is not necessary to specify this attribute, as connections are efficiently managed internally. However, for connections between different hosts on the network, it is vital to properly establish and adjust the bandwidth according to the characteristics of the multimedia stream.

3.4.2 destination

Defines the properties corresponding to the destination point, indicating where the multimedia stream will be received. Table 17 presents the attributes of *destination*.

3.5 Node Types

3.5.1 virtualLink

The interconnections between two elements (virtual device or VNF) of a *multimedia SFC* are described by *virtualLink*. This element provides information on two other components, the source point (*source*) from which the stream is sent and the destination point (*destination*) where the stream will be received. Table 18 presents the attributes of *virtualLink*.

3.5.2 vms

A *vms* or *multimedia VNF* is used to process one or multiple multimedia streams and generate one or more output results. The result of a *multimedia VNF* can be sent to one or multiple destinations, including an end-user application. Table 19 presents the attributes of *vms*.

Table 17. Attributes of *destination*.

Name	Description
destinationName* **Type:** *String*	Which component of the *multimedia SFC* will receive the stream. This data must be identical to the one configured in a *vmsName* of one of the *multimedia VNF* used in the same *multimedia SFC*.
address **Type:** *mPrism .datatype .addressData*	Complex data type initially defined by TOSCA-NFV and adapted in L-PRISM, which describes a network address. This property is defined using the *addressData* data type, as detailed in Section 3.1.2.
port* **Type:** *mPrism .datatype.port*	Describes the port where the multimedia stream will be received. This property is defined using the port data type, as detailed in Section 3.1.9.
inputType* **Type:** *String*	Describes the type of multimedia stream that the destination of the connection accepts.

Table 18. Attributes of *virtualLink*.

Name	Description
source* **Type:** *mPrism .relationshipType .source*	The source point of the multimedia stream.
destination* **Type:** *mPrism .relationshipType .destination*	The destination point of the multimedia stream.

3.6 chainModel

The *chainModel* presents the attributes that describe a *multimedia SFC*. Table 20 presents the attributes of *chainModel*.

Finally, Table 21 provides a comparative overview of the M-PRISM metamodel in relation to TOSCA-NFV and ETSI-NFV. The comparison focuses on four key aspects: the domain and focus of each approach, their capabilities for modeling connectivity and networking, the additional functionalities they provide, and their support for virtualization. This comparison emphasizes the distinctive features of M-PRISM, particularly its suitability for specifying multimedia applications.

4 L-PRISM

This section presents L-PRISM, a DSL tailored to describe *multimedia SFCs* based on *multimedia VNFs*. L-PRISM is mainly based on the architecture proposed in [Battisti *et al.*, 2020c] and the M-PRISM metamodel described in Section 3. Our proposed DSL was developed following the stages presented in Negm *et al.* [2019]. Based on this approach, the development of a DSL involves five key stages:

Table 19. Attributes of *vms*.

Name	Description
vmsName* **Type:** *String*	The name of a *multimedia VNF* must be unique in a *multimedia SFC*. This data type works as an *ID* in a *virtualLink*.
vmsDescription **Type:** *String*	*Multimedia VNF* description.
vmsType* **Type:** *mPrism .artifactType .swImage*	The virtualized images used for creating the *multimedia VNF*. This property is defined using the *swImage* artifacts type, as detailed in Section 3.3.1.
startup-Parameters **Type:** *mPrism .dataType .configurable Properties*	Data type defined initially by TOSCA-NFV; it describes the initial configuration properties of the *multimedia VNF*. This property is defined using the *configurableProperties* data type, as detailed in Section 3.1.8.
host* **Type:** *mPrism .artifactType .host*	Network node that will host the *multimedia VNF*. The list of available nodes must be shared with the developer of the *multimedia SFC*. This property is defined using the *host* artifact type, as detailed in Section 3.3.2.
virtualCompute **Type:** *mPrism .capabilitiesType .virtualCompute*	Computational properties assigned to the *multimedia VNF*. This property is defined using the *virtualCompute* capabilities type, as detailed in Section 3.2.2.

Table 20. Attributes of *chainModel*.

Name	Description
chainName* **Type:** *String*	Name of the *multimedia SFC*.
chainDescription **Type:** *String*	Description of the *multimedia SFC*.
devices* **Type:** *list*	List of virtual devices that compose the *multimedia SFC*. The elements of this property are defined using the *device* artifact type, as detailed in Section 3.3.3.
vmss* **Type:** *list*	List of *multimedia VNFs (vms)* that compose the *multimedia SFC*. The elements of this property are defined using the *vms* node type, as detailed in Section 3.5.2.
virtualLinks* **Type:** *list*	List of virtual links between *device* and *multimedia VNF* (*vms*). The elements of this property are defined using the *virtualLink* node type, as detailed in Section 3.5.1.

- **Domain Analysis**: This stage focuses on understanding the target domain, identifying its core concepts and relationships, and clearly defining its boundaries. We present the L-PRISM Domain

Table 21. Comparison of key aspects between M-PRISM, TOSCA-NFV, and ETSI-NFV.

Aspect	M-PRISM	TOSCA-NFV	ETSI-NFV
Domain and Focus	Specialized in multimedia applications based on virtualization. It focuses on the description of *multimedia SFCs* and the integration of multimedia sensors.	Focused on the modeling and orchestration of VNFs according to NFV standards, without multimedia specialization.	Centered on the packaging, metadata, and structural definition of VNFs to ensure interoperability and validation. It does not model multimedia services or flows.
Additional Functionalities	(i) Introduces exclusive artifacts, such as the *host* artifact (*lPrism.artifacts.vms.host*) for defining nodes with special properties (e.g., *isMaster*). (ii) Defines multimedia-specific data types (e.g., *virtualGraphicsCard*, and multimedia *ports* with defined types, formats, and quality parameters). (iii) Offers tailored configuration options for processing and transmitting multimedia streams.	Provides generic data types for addressing, connectivity, and compute nodes, but lacks multimedia-specific constructs (e.g., no multimedia host or graphical artifacts).	Does not support multimedia modeling. Its scope is limited to the structure, metadata, and integrity validation of VNF packages, following ETSI specifications.
Connectivity and Networking Modeling	Supports detailed modeling of connectivity between *VDs* and *multimedia VNFs*, accounting for multimedia characteristics (e.g., stream types, input/output ports, formats, and quality).	Provides generic connectivity constructs (e.g., *virtualLink*, VDU) applicable to VNFs, but without multimedia-specific semantics.	Does not model connectivity; focuses instead on the internal structure and metadata of VNF packages.
Support for Virtualization	Designed to address the specific requirements of *multimedia VNFs* and *multimedia SFCs*, reducing configuration complexity and promoting interoperability in multimedia environments.	Targeted at generic virtualization of network functions, with no dedicated support for multimedia stream management.	Oriented toward the packaging and validation of VNFDs, without mechanisms for modeling virtualized multimedia functions.

Analysis in Section 4.1.

- **Design:** Based on the domain analysis, this stage involves designing the DSL's structure and features to effectively express the problems and solutions specific to the domain. We present the L-PRISM Design Analysis in Section 4.2.
- **Implementation:** The DSL is then implemented using appropriate tools or frameworks, resulting in a concrete, usable language. We present the L-PRISM implementation in Section 4.3.
- **Evaluation:** In this stage, the DSL is assessed to determine whether it meets the intended business or technical requirements. We present the results of the L-PRISM evaluation in Section 6.
- **Maintenance:** This phase ensures the DSL remains relevant over time. It involves updating the language to reflect changes in the domain, introducing new syntax or semantics, improving tooling support, and incorporating user feedback. However, this paper does not address the maintenance phase, as it focuses on the proposal of a new language.

4.1 L-PRISM Domain Analysis

Analyzing multimedia applications in virtualized environments is essential to identify and abstract the characteristics that influence their proper functioning. In the context of building the DSL proposed in this work, the analysis focuses on configurable aspects and non-functional requirements, such as computational resource allocation, network requirements, and communication needs, while excluding the internal implementation details of the applications. Multimedia applications are typically structured as chains of functions that process multimedia flows in a predefined sequence, similar to a directed graph. This chaining, referred to in this work as a Multimedia Service Function Chain (*multimedia SFC*), consists of Multimedia Virtual Network Functions (*multimedia VNFs*), which represent the virtualization of individual multimedia functions. Virtualization enables these functions to be decomposed into specific layers, establishing a modular and flexible foundation for designing *multimedia SFCs*.

The use of virtualization applied to multimedia applications has been extensively studied, and several of these works have served as the foundation for the domain analysis in our research. For instance, in [Battisti *et al.*, 2020c], *multimedia VNFs* are referred to as Virtual Multimedia Sensors (VMS), highlighting their implementation in edge and cloud environments. In [Imagane *et al.*, 2018], system segmentation is explored using virtualization technologies, allowing the identification of specific configurations relevant to our analysis. Similarly, platforms like OpenStack, used in [Alvarez *et al.*, 2019] and [Imagane *et al.*, 2018], provided insights into the management of computational and network resources, which are fundamental aspects for the design of *multimedia SFCs*.

A notable proposal is the V-PRISM architecture [Battisti *et al.*, 2020c], designed for edge-cloud environments and divided into three levels: Devices (Things), Edge, and Cloud. This architecture addresses the needs of *multimedia SFCs* at two levels:

- Transmission resources: These include latency, CPU usage, available memory, and the resource usage history of edge nodes;
- Application-specific resources: These involve bandwidth and computational capacity required by each *multimedia VNF*.

Our analysis also revealed common patterns in *multimedia SFCs*. For instance, [Alvarez *et al.*, 2019] introduces a teleimmersive 3D application that connects players and spectators through functions such as the 3D Multimedia Transformer (T3D) and the Replay function. When virtualized, these components become vT3D and vReplay, forming a *multimedia SFC* composed of both physical and virtual components interconnected by Virtual Links (VL). Other examples include:

- Augmented Reality [Manias *et al.*, 2024]: Comprising four *multimedia VNFs*: Interest Point Detection (IPD), Descriptor Generation (DG), Image Description Database (IDD), and Content Matcher (CM);
- Live Streaming [Alvarez *et al.*, 2019; Manias *et al.*, 2024]: Consisting of five *multimedia VNFs*: Source Encoder (SE), Live Video Processor (LVP), Organizer/Packer (OP), Media Optimizer (MO), and Virtual Content Delivery Network (vCDN).

The results of our analysis identified the following key requirement for L-PRISM:

a) **Multimedia Virtual Network Functions (*multimedia VNFs*):**
   - Primarily based on the Virtual Multimedia Sensors (VMS) described in [Battisti *et al.*, 2020c], *multimedia VNFs* encapsulate individual multimedia functions within a modular design, facilitating their deployment in edge and cloud computing environments;
   - Additionally, different types of *multimedia VNFs* were identified, notably Virtual Devices (VD) [Battisti *et al.*, 2020c]. Unlike other types of *multimedia VNFs*, these replicate the multimedia stream from their physical counterparts (Physical Devices, PD) as needed, making them available as services for multiple *multimedia SFCs*.

b) **Resource Descriptors:**
   - Derived from the characteristics and resource allocation strategies explored in [Imagane *et al.*, 2018] and [Alvarez *et al.*, 2019], these descriptors enable the specification of the computational and network requirements for each *multimedia VNF* within a *multimedia SFC*;
   - Our analysis also highlights that, unlike traditional applications, multimedia applications can process a single type of multimedia stream at different quality levels. This means that a single *multimedia VNF* may have variable resource allocation requirements depending on the quality of the stream being processed.

c) **Virtual Links (VL):**
   - Adapted from the connectivity models described in [Alvarez *et al.*, 2019; Battisti *et al.*, 2020c], Virtual Links (VL) represent the interactions between *multimedia VNFs* within a *multimedia SFC*. These links provide a structured framework to define the dependencies and relationships between components.

d) **Application Templates:**
   - Based on common patterns identified in [Manias *et al.*, 2024] and [Alvarez *et al.*, 2019], these templates simplify the definition of *multimedia SFCs*. They also provide a global view of the structure and configuration of a *multimedia SFC*, reducing the complexity of its development and management.

Finally, our analysis identified that virtualization and modular segmentation are key elements for designing and managing multimedia applications in virtualized environments. These strategies enable efficient resource management and facilitate integration with advanced technologies such as SDN (Software Defined Networking) and edge-cloud architectures like V-PRISM. Furthermore, the study of *multimedia SFCs* reveals common patterns and replicable approaches, underscoring the importance of adopting standardized and flexible models to address the challenges of these applications. This understanding lays the groundwork for developing our DSL, capable of efficiently modeling and managing *multimedia SFCs*.

4.2 L-PRISM Design

Based on the analysis conducted in the previous section and considering our primary goal of creating *multimedia SFCs* based on *multimedia VNFs*, the design of our DSL is grounded in the M-PRISM metamodel and the V-PRISM architecture [Battisti *et al.*, 2020c]. Building upon these foundations, the DSL supports the description and implementation of *multimedia SFCs* in distributed environments, primarily composed of edge and cloud nodes that adopt lightweight virtualization technologies, such as container-based platforms. The design process of our DSL consists of the following steps.

4.2.1 Definition of *multimedia SFC* Components

In this step, we identify the main components that constitute *multimedia SFCs*. Figure 3 provides an example and outlines the primary components of a *multimedia SFC*:

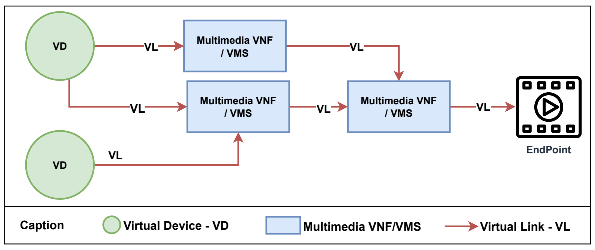


Figure 3. Example of a *multimedia SFC*.

a) ***Virtual Device (VD):*** Represents the virtualized version of a physical multimedia device;
b) ***Multimedia VNFs/VMS:*** Functions responsible for processing multimedia flows. Unlike VDs, *multimedia VNF* instances are created exclusively for each *multimedia SFC*;
c) ***Virtual Links (VL):*** Provide communication between (i) VDs and *multimedia VNFs*, (ii) different *multimedia VNFs*, and (iii) *multimedia VNFs* and endpoints (user applications).

4.2.2 Design of Virtualized Component

The design of this component is inspired by the Proof of Concept (PoC) of the V-PRISM architecture and the L-PRISM metamodel. Since Docker is used as the lightweight virtualization technology in the PoC, the characteristics of VDs and *multimedia VNFs* are defined based on the structure of the *device* and *vms* components in the M-PRISM metamodel.

4.2.3 Design of Communication Between Virtualized Components

The communication design between VDs and *multimedia VNFs* is also based on the PoC of the V-PRISM architecture and the L-PRISM metamodel. For configuring communication, we use the *Virtual Link* component from the M-PRISM metamodel.

4.2.4 Design of Virtualized Components Orchestration

The design of virtualized component orchestration addresses the management of the lifecycle and interactions of components within *multimedia SFCs*. Key orchestration considerations identified include:

a) ***Virtual Devices (VDs):*** These components must already be created and treated as resources within the edge or cloud. Each VD should have an address (IP or MAC) and an identifier to retrieve information or subscribe to the services it provides;
b) ***Multimedia VNFs:*** These should be created following a specific rule: the *multimedia VNF* being instantiated must know the IP addresses of the *multimedia VNFs* to which it will send the processed multimedia flows. Therefore, a *multimedia SFC* must be constructed from the endpoint backward;
c) ***Endpoints:*** The points where multimedia flows will be rendered must support the necessary technologies to capture and reproduce these flows.

4.2.5 Validation and Testing Design

The validation and testing design is presented in detail in Section 6. For those tests, the *multimedia VNFs* available from ALFA [Battisti *et al.*, 2020c] were adapted, and its back-end and front-end were upgraded to a new version called ALFA 2.0, which is described in more detail in Section 5.

4.3 L-PRISM Implementation

The implementation stage of our DSL employs a compilation approach, as we primarily focus on defining a high-level language that is subsequently translated into a low-level language for compilation. This stage considers three key aspects, as defined by Negm *et al.* [2019], which are described as follows:

4.3.1 Language Structure

To define the language structure of our DSL, we adopted a model-based approach, built upon the analysis and design described in the previous stages and leveraging the M-PRISM metamodel.

The following describes the attributes that compose a *multimedia SFC* and the key elements that define its structure.

Source Code 1 gives a simple example of a *multimedia SFC* described using L-PRISM. Figure 4 shows the visual representation of that SFC. It is composed by one *device*, one *multimedia VNF* and two *virtualLink*. It has one video source (VD 2) that is handled by a VNF (*Multimedia VNF 1*) that sends the video stream to the final application (*IP: 192.168.0.101, Port: 1002*).

4.3.2 Language Editor

L-PRISM uses YAML, a widely adopted data serialization language known for its simplicity, readability, and compatibility with a broad range of modern tools [YAML, 2021]. YAML provides a clear and easily interpretable structure, making it an ideal choice for defining and configuring *multimedia SFCs*. Furthermore, YAML allows specifications to be written in any text editor, such as Visual Studio Code or Notepad++, without the need for specialized tools.

4.3.3 Language Semantics

To define the semantics of L-PRISM, we adopted the translational semantics approach, commonly used in DSL design. Programs written in L-PRISM are first represented

Source Code 1. A multimedia SFC described in L-PRISM.

```
chainModel:
    chainName: multimedia SFC test
    chainDescription: Video Flux
    devices: #List of devices
    - #In this example 1 device
        deviceName: VD 2
        deviceId: 638e70b10a1fbd0026fccaf4
    vmss: #List of multimedia VNFs
    - #In this example 1 multimedia VNF
        vmsName: Multimedia VNF 1
        vmsDescription: Multimedia VNF 1
            description
        vmsType: 638e70b10a1fbd0026fccae8
        host: 192.168.0.117 # IP
        virtualCompute:
          virtualMemory:
              size: 1024
          virtualCPU:
              numCpu: 1
    virtuallinks: # List of Virtual Links
    - #virtualLink 1 ("VD 2" -> "Multimedia VNF
        1")
        source:
          sourceName: VD 2
          sourceType: device
          outputType: video
        destination:
          destinationName: Multimedia VNF 1
          address: #define for orchestrator
          port: 5000 # port of multimedia VNF
          inputType: video
    - #virtualLink 2 (final connection)
        source:
          sourceName: Multimedia VNF 1
          sourceType: vms
          outputType: video
        destination:
          destinationName: 192.168.0.101
          address: 192.168.0.101
          port: 10002
          inputType: video
```

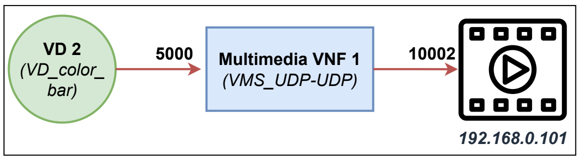


Figure 4. Multimedia Chain example.

in a YAML file for simplicity and interoperability. This file is processed to translate the specification into a lower-level programming language (e.g., JavaScript, Java, or C++), enabling integration with broader systems. Following a classical compiler-inspired approach, an Abstract Syntax Tree (AST) is constructed by traversing and analyzing the YAML file.

5 ALFA 2.0: Proof of Concept

This section describes the integration of L-PRISM into a platform named ALFA [Battisti *et al.*, 2020c,a][2]. Our PoC named *ALFA 2.0* increases the capabilities of the previous version by enabling the execution of SFCs using our proposed L-PRISM language. The new implementation is released under the MIT License and the source code is available on GitHub[3].

5.1 Logic Components

Our proposed architecture comprises a series of interconnected logical components, as depicted in Figure 5. These components manage the processing of multimedia streams generated by multimedia devices and consumed by IoMT applications. In this paper, we focus on the components introduced in this new version. A comprehensive description of all components of the architecture is available in [Battisti *et al.*, 2020b; Quico *et al.*, 2023].

IoMT applications are generally deployed in the cloud, while physical multimedia devices are typically located in the Things tier. Stream processing is handled by a set of *multimedia VNFs* provided and hosted by the architecture. Both *multimedia VNFs* and all ALFA 2.0 components can be distributed across nodes within the edge-cloud continuum [Maia *et al.*, 2024]. ALFA 2.0 leverages containers to run VNFs and virtual devices, allowing language independence and seamless integration of solutions from various vendors.

The new components introduced in ALFA 2.0 are *L-PRISM Interpreter*, *L-PRISM Executor* and the *Chain Model*. The *L-PRISM Interpreter* performs both semantic and syntactic analysis on the *multimedia SFCs* described using the L-PRISM language. Meanwhile, the *L-PRISM Executor* interacts with other components within the architecture, facilitating the instantiation of *multimedia VNFs* needed to execute the requested *multimedia SFC*.

Another key component is the Chain Model, which stores information about the sequence of *multimedia VNFs* that compose the *multimedia SFC*. It keeps details about each *multimedia VNF* instance such as the execution node and the connection to the next VNF in the chain. Additionally, it also stores data about the resource usage for each VNF instance.

To facilitate the interaction with the platform, the ALFA implementation includes an HTTP API component that enables the execution of *multimedia VNFs*. However, the first version of the API did not support *multimedia SFCs*, it was only capable of executing single *multimedia VNFs*. To address this restriction, a new endpoint was introduced in ALFA 2.0. This new endpoint receives and processes a YAML file containing the *multimedia SFC* described using the L-PRISM language.

5.2 *Multimedia SFC* Chain Creation

Multimedia SFCs are complex applications composed of multiple *multimedia VNFs*. Before processing a multimedia stream within the SFC, all *multimedia VNFs* must be executed and linked to the next one in the chain. ALFA 2.0 provides the necessary components to seamlessly execute the requested *multimedia SFC* based on a descriptor file created using the L-PRISM language.

[2]Source code available at https://github.com/midiacom/alfa

[3]Source code of ALFA 2.0 https://github.com/fventuraq/alfa

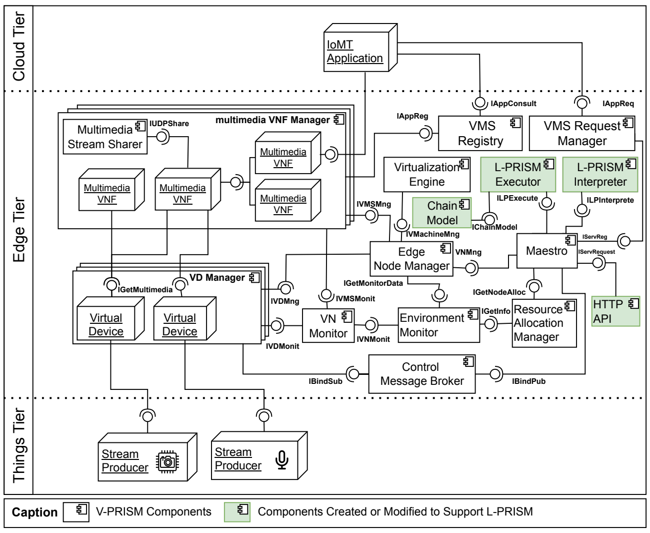


Figure 5. L-PRISM Logic Architecture.

The initialization of a *multimedia SFC* consists of two phases. The first phase is the admission phase, where the L-PRISM Interpreter checks if the requested YAML file is well-formed both syntactically and semantically. The second phase involves creating the necessary components to execute the requested *multimedia SFC*. Figure 6 presents a sequence diagram that illustrates the process of creating a *multimedia SFC*.

5.3 Web Portal

The ALFA API enables the interoperability between various softwares and the ALFA components. Building on the existing ALFA implementation, which already has a Web Portal to manage *multimedia VNFs*, we extend its functionality by adding new capabilities. These new features enable users to upload YAML files containing *multimedia SFC* descriptions written in L-PRISM language. This extension integrates seamlessly with the ALFA platform, allowing efficient operation and orchestration of *multimedia SFCs* directly through the web browser.

Figure 7(A) illustrates the Web Portal interface used to upload L-PRISM files containing *multimedia SFC* descriptions. The Web Portal displays information about the entities managed by the system, for example, available VNF Types in each node, Virtual Devices in execution, and available nodes within the environment. Through the Web Portal, it is possible to manage several *multimedia SFCs* with VNFs executed in multiple edge-cloud nodes.

Figure 7(B) illustrates the output of a complex *multimedia SFC* described using L-PRISM. The SFC is composed of multiple VNFs that perform tasks such as image color conversion and video combination. After being processed, the multimedia stream is delivered to the final user and displayed in the IoMT application. Figure 8 presents the *multimedia SFC* whose output is illustrated in Figure 7(B).

In the Web Portal we also implemented an interactive tool to customize the *multimedia SFC* deployed with L-PRISM. After uploading the file containing the *multimedia SFC* description, users can modify the *multimedia VNFs* within the chain and adjust other characteristics of the *multimedia SFC*. These changes can be applied in real-time to the SFC currently in execution or exported in YAML format for future deployment.

5.4 *Multimedia SFC* Placement

The SFC placement is the process of identifying a feasible way to allocate VNFs within an SFC to meet various requirements related to performance, cost, and policy compliance [Santos *et al*., 2022a]. This problem has been addressed through various approaches, which can be categorized into centralized and decentralized methods [Macedo *et al*., 2023]. In a centralized approach, a single orchestrator is responsible for the global view and control of the entire network, making placement decisions based on a complete understanding of the available resources and the network topology [Battisti *et al*., 2022]. The decentralized approach distributes the decision-making process across multiple nodes or domains, each making local placement decisions based on partial knowledge of the network and computational resources [Rodrigues *et al*., 2023; Huff *et al*., 2019]. L-PRISM is agnostic to placement algorithms and can be utilized by systems that employ centralized or decentralized approaches.

In ALFA 2.0, we addressed the *multimedia SFC* placement problem using a centralized approach. In Figure 6, the Resource Allocation Manager component is responsible for creating the placement plan to execute the *multimedia SFC* requested. This component implements a Greedy approach, selecting the node with the lowest resource usage to execute the *multimedia VNFs*. It is important to note that we decoupled the SFC placement decision from the execution of the *multimedia SFC*, allowing new approaches to be implemented without requiring changes in the other architectural components.

In this section, we presented our proof of concept ALFA 2.0, which extends a previous version of ALFA. The main contribution is the added capability to execute *multimedia SFCs* described using L-PRISM. In the following section, we will present experimental results that demonstrate the feasibility and effectiveness of using L-PRISM to describe *multimedia SFCs*.

6 L-PRISM Qualitative Analysis

In this section, we evaluate the creation of *multimedia SFCs* using multiple techniques [Rodrigues *et al*., 2019; Koziolek, 2008]. The experiment was carried out with computer science students to assess the efficiency of using L-PRISM compared to a traditional approach. The experiments were modeled using the Goal Question Metric (GQM) methodology [Koziolek, 2008].

6.1 Goal Question Metric

The experiment aims to evaluate the development of *multimedia SFCs* using multiple techniques. The GQM methodology was adopted to guide our experimental phase. Table 22 presents our goals.

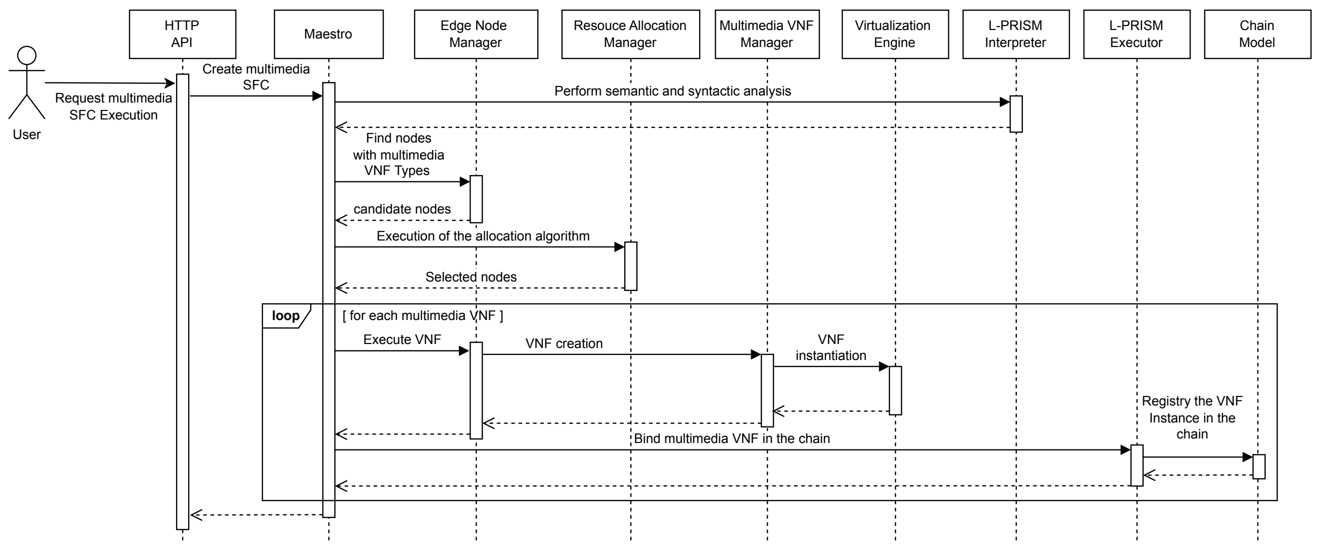
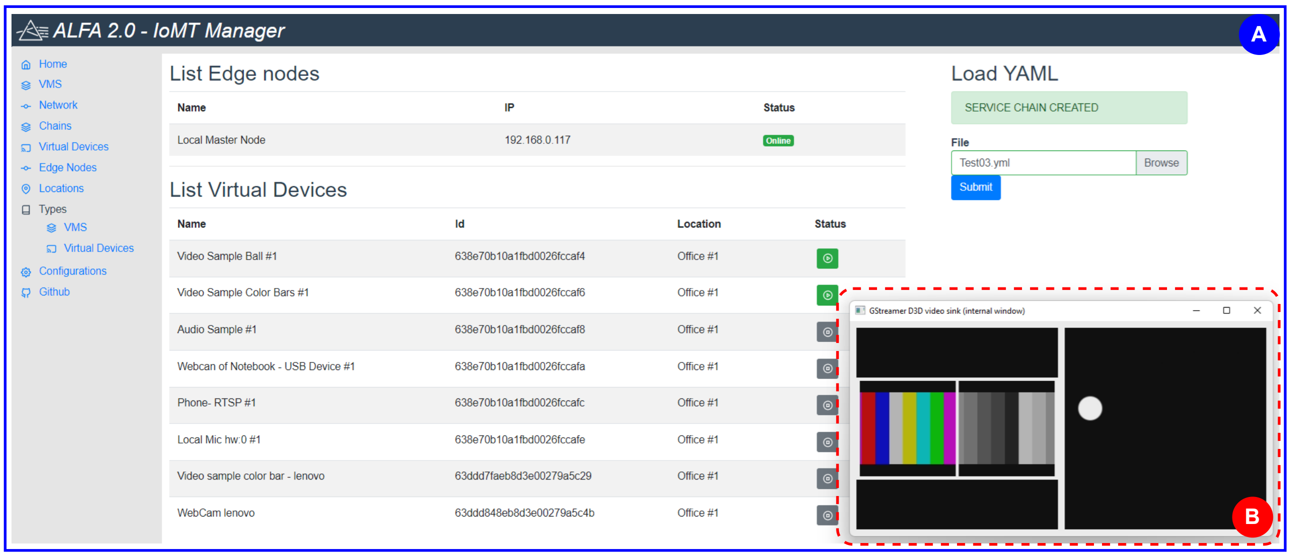

HTTP API
Maestro
Edge Node Manager
Resouce Allocation Manager
Multimedia VNF Manager
Virtualization Engine
L-PRISM Interpreter
L-PRISM Executor
Chain Model
User
Request multimedia SFC Execution
Create multimedia SFC
Perform semantic and syntactic analysis
Find nodes with multimedia VNF Types
candidate nodes
Execution of the allocation algorithm
Selected nodes
loop
[for each multimedia VNF]
Execute VNF
VNF creation
VNF instantiation
Registry the VNF Instance in the chain
Bind multimedia VNF in the chain

Figure 6. *Multimedia SFC* sequence diagram creation.



List Edge nodes

Name	IP	Status
Local Master Node	192.168.0.117	Online

List Virtual Devices

Name	Id	Location	Status
Video Sample Ball #1	638e70b10a1fbd0026fccaf4	Office #1	
Video Sample Color Bars #1	638e70b10a1fbd0026fccaf6	Office #1	
Audio Sample #1	638e70b10a1fbd0026fccaf8	Office #1	
Webcan of Notebook - USB Device #1	638e70b10a1fbd0026fccafa	Office #1	
Phone- RTSP #1	638e70b10a1fbd0026fccafc	Office #1	
Local Mic hw:0 #1	638e70b10a1fbd0026fccafe	Office #1	
Video sample color bar - lenovo	63ddd7faeb8d3e00279a5c29	Office #1	
WebCam lenovo	63ddd848eb8d3e00279a5c4b	Office #1	

Load YAML

SERVICE CHAIN CREATED

File

Test03.yml Browse

Submit

Figure 7. (A) Web Portal for executing L-PRISM files in ALFA 2.0. (B) Multimedia stream processed through the created SFC.

Table 22. Evaluation goals.

Goal	Description	Perspective
G1	Analyze the application engineering process with and without L-PRISM to evaluate the efficiency and productivity of developing *multimedia SFC*.	Efficiency and Productivity
G2	Evaluate the comprehensibility of L-PRISM to analyze if variables, attributes, and structures are understandable for the participants.	Usability

6.1.1 G1 - Questions and metrics

For *G1*, six questions related to efficiency and productivity in the process of developing *multimedia SFCs* were raised. Table 23 presents the G1 questions and the metrics related to them.

The questions for *G1* were adapted from the Technology Acceptance Model (TAM) [Surendran, 2012], a theoretical framework commonly used to assess user attitudes toward technology adoption using multiple acceptance-related dimensions. The goal was to quantitatively evaluate the *multimedia SFC* development process using L-PRISM and the traditional method. Metrics *M2, M3, M4* and *M5* were measured using the Likert scale (1 - strongly disagree to 5 - strongly agree), and for metrics *M1* and *M6*, the time in minutes required to perform each task was requested.

6.1.2 G2 - Questions and metrics

For goal *G2*, seven questions *(Q1-Q7)* related to Cognitive Dimensions of Notations (CDN) [Blackwell and Green, 2000] were used. This approach helped us evaluate L-PRISM from a usability point of view. Table 24 presents the proposed questions for *G2* and the respective metrics. Those questions should be answered using the Likert scale, for *Q3* and *Q4* (1 - strongly disagree to 5 - strongly agree) and for *Q1, Q2, Q5, Q6* and *Q7* (1 - very difficult to 5 - very easy).

Table 23. Questions and metrics for goal *G1*.

#Q	Description	Metrics
Q1	Is the application engineering process using L-PRISM effective in terms of time for developing *multimedia SFC*, compared to the traditional approach (ALFA)?	M1 - Development effort
Q2	Does the developer claim that using L-PRISM makes it easier to understand the functional and non-functional requirements of the *multimedia SFC*?	M2 - Understanding of requirements
Q3	Does the developer claim that using L-PRISM helps create *multimedia SFC*?	M3 - Perceived ease of use
Q4	Does the developer claim that L-PRISM is useful to create *multimedia SFC*?	M4 - Perceived utility
Q5	Does the developer claim that using L-PRISM makes it easier to reuse the *multimedia SFC* created with L-PRISM to create new *multimedia SFC*?	M5 - Perceived reuse
Q6	Is the process of modifying *multimedia SFC* faster with L-PRISM compared with the traditional method (ALFA)?	M6 - Reuse effort

Table 24. Questions and metrics for goal *G2*.

#Q	Description	Metrics
Q1	How easy is it to visualize or find the different elements and attributes of L-PRISM when creating or changing a *multimedia SFC*?	M1 - Visibility
Q2	How easy is modifying a *multimedia SFC* with L-PRISM?	M2 - Viscosity
Q3	Is the L-PRISM language too verbose to specify a *multimedia SFC*?	M3 - Diffuseness
Q4	How well do the L-PRISM elements and attributes represent a *multimedia SFC*?	M4 - Closeness of Mapping
Q5	How easy is it to understand the structures and data types of L-PRISM?	M5 - Role Expressiveness
Q6	There are structures and data types in L-PRISM that can be closely related, and changes in one can affect the other. Are these dependencies easy to be seen?	M6 - Hidden dependencies
Q7	Does L-PRISM generally seem easy or difficult to understand (for example, when changing different elements of a *multimedia SFC*)?	M7 - Hard mental operations

6.2 Procedure

The evaluation of L-PRISM was divided into two phases. The first phase involved training participants to understand and create *multimedia SFCs* using both L-PRISM and the alternative method. In the second phase, the designed tasks addressed the questionnaires for goals G1 and G2.

6.2.1 Training phase

In this phase, participants were trained in two topics:

- **Understanding and development of *multimedia SFCs* using *ALFA***: Participants were asked to manually implement a *multimedia SFC* using the original *ALFA* platform [Battisti *et al.*, 2020c]. This aimed to simulate the development of *multimedia SFCs* using the traditional method, where each component of the *multimedia SFC* is manually executed with *ALFA*;
- **Development of *multimedia SFCs* using L-PRISM**: For this, *ALFA 2.0* was used, which supports uploading a *multimedia SFC* described with L-PRISM (YAML format file).

Additional material was also prepared and available on the L-PRISM website[4]. This site contains information about L-PRISM and various examples of how to implement *multimedia SFCs* using both L-PRISM and the traditional method employed in these tests.

The types of components available for this experiment were two virtual devices. The first device produces a video of a white sphere bouncing on a black box, which we called *VD_video_ball*, and the second device produces a video of colored bars, which we called *VD_color_bar*. Three types of *multimedia VNFs* were also made available, namely (i) a *multimedia VNF* that transforms a color video to grayscale called *VMS_gray*, (ii) a *multimedia VNF* that only transfers UDP-UDP data and does not apply any changes to the original stream, called *VMS_udp*, and (iii) a *multimedia VNF* that merges video-type media streams, grabbing two multimedia streams and transforming them into one, called *VMS_merge*. The tasks are described in the following section.

6.2.2 Execution phase

For the execution phase, different components (*swImage*) were made available to complete the four proposed tasks. These components included two virtual devices. The first device generates a video of a white sphere bouncing inside a black box, referred to as *VD_video_ball*, and the second device generates a video of colored bars, referred to as *VD_color_bar*. Additionally, three types of *multimedia VNFs* were provided: (i) a *multimedia VNF* that transforms a color video into grayscale, called *VMS_gray*; (ii) a *multimedia VNF* that transfers data from UDP to UDP without modifying the original stream, called *VMS_udp*; and (iii) a *multimedia VNF* that merges video streams, combining two multimedia streams into a single output, called *VMS_merge*. The tasks were:

- **T1**: This task is to develop a *multimedia SFC*, where *VMS_gray* will need to subscribe to the *VD_color_bar* to process the multimedia stream and publish the result to a point within the network (*IP* and *PORT*);

[4]https://fventuraq.github.io/lprism.html

- **T2**: This task compares a raw multimedia stream with the same processed multimedia stream. To perform this task, it will be necessary to use *VMS_merge* that will receive the multimedia stream processed by *VMS_gray* and subscribe to *VD_color_bar* to receive the original stream, *VMS _gray* will need to subscribe to *VD_color_bar* in order to process this multimedia stream and send it to *VMS_merge*. Finally, *VMS_merge* will publish the result at a network point (*IP* and *PORT*);
- **T3**: This task consists of adding the *VD_video_ball* multimedia stream to *T2*. It is necessary to create a new *VMS_merge* that will receive the result of *T2* through one of its ports, and the other port will subscribe to *VD_video_ball*. Finally, it will publish its result on a given network point (*IP* and *PORT*). The visual result of *T3* is presented in Figure 7(B), while Figure 8 illustrates the corresponding *multimedia SFC*;
- **T4**: This task consists of replicating *T3*, changing the final destination port to where the final stream is sent. The idea is to be able to visualize two identical *multimedia SFCs*, created in *T3* and *T4*, running at the same time.

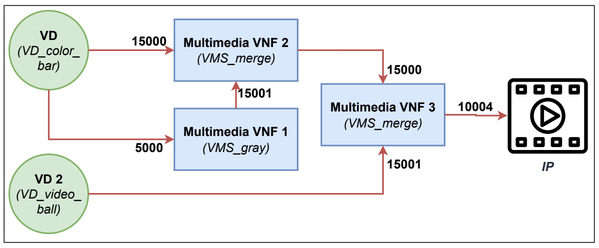

VD (VD_color_bar)
15000
Multimedia VNF 2 (VMS_merge)
15000
15001
Multimedia VNF 3 (VMS_merge)
10004
5000
Multimedia VNF 1 (VMS_gray)
15001
VD 2 (VD_video_ball)
IP

Figure 8. Diagram of the expected *multimedia SFCs* described in *Task 3*.

6.3 Participants

In order to carry on user experiments, the consent of each participant was requested through a Free and Informed Consent Form (FICF) and all collected data was anonymized. All responses were collected through a Google Forms questionnaire. No personal or sensible data was collected from participants, except their gender and age. Computer science students were invited to participate. Fifteen participants aged 20-39 years accepted our invitation. They were 14 men and 1 woman. Their academic degree was: seven were undergraduate, three were graduate, and five post-graduate students. We asked their level of experience on XML, JSON and YAML (from 1 (no experience) to 5 (a lot of experience)). The median answers were 4 for XML, 5 for JSON, and 3 for YAML. We built a testbed with edge nodes in our lab to run the experiments. Eight participants did it physically in the lab and seven participants did it remotely. All of them were observed all the time during the experiment, even the remote ones.

6.4 Analysis of Results

This section presents a detailed analysis of the evaluation results to determine whether L-PRISM meets its objectives of efficiency, productivity, and usability. The evaluation was structured around two main goals: *G1*, which focuses on the productivity and efficiency of L-PRISM, and *G2*, which examines its usability through cognitive dimensions of notations. The subsequent subsections provide a detailed discussion of the results and their implications.

6.4.1 G1 - Evaluation

Questions *Q2* to *Q5* of G1 are related to the productivity of L-PRISM. Figure 9 shows that the results of these questions are positive and the responses expected to validate L-PRISM were positive. With this, we can conclude that the *G1* goal was achieved.

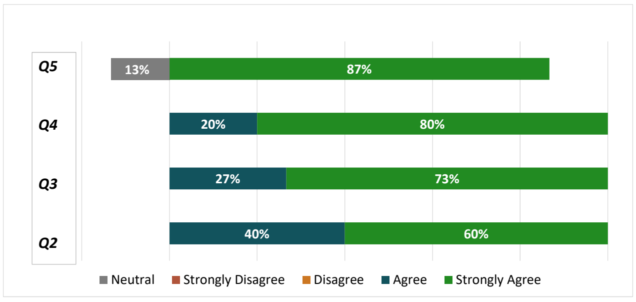


Figure 9. Participant Responses for Questions Q2-Q5 of G1.

Questions *Q1* and *Q6* are related to efficiency. To answer *Q1*, Figure 10 shows the average time in minutes it took for participants to complete each task with both methods (L-PRISM and Traditional), with a confidence interval of 90%. It can be seen that Tasks 1, 2, and 3 have similar times, although the learning curve to learn a new language is much greater than the use of an intuitive interface, L-PRISM is equivalent to the traditional method for new developers of *multimedia SFCs*. However, when analyzing the relative values within L-PRISM, it is observed that Task 2 took on average *7%* more time than *Task 1*, and *Task 3* took *27%* more than *Task 2*, reflecting the increasing complexity between tasks. Most notably, *Task 4*, despite having similar complexity to *Task 3*, shows a *78%* reduction in time compared to *Task 2*, suggesting a training effect. This result indicates that L-PRISM enables developers to achieve efficiency comparable to or even superior to the traditional method as they become more familiar with the language.

In the case of Task 4, L-PRISM has proven to be superior than the traditional method. This is because developers came to understand L-PRISM better with the previous tasks, and most developers perceived that they could reuse the code from Task 3 for Task 4. This task also answers *Q6* of *G1*, which focuses on evaluating if L-PRISM allows the reuse of an already created *multimedia SFC*. As it can be seen in Figure 10, using L-PRISM, participants completed Task 4 in an average time 77.8% shorter than with the traditional method, due to L-PRISM code reuse.

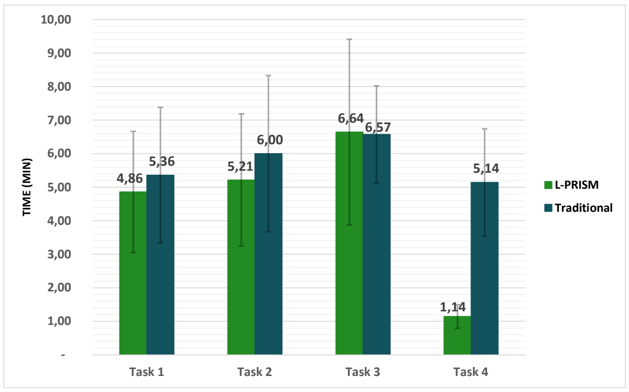


Figure 10. Time needed per task in L-PRISM and Traditional methods.

6.4.2 G2 - Evaluation

The answers to the G2 questions related to the cognitive dimensions are presented in Figure 11. It should be noted that, except for *Q3* about language diffuseness (verbosity), where a negative response was expected, all the others had positive feedback from the subjects. As it can be seen, the responses for Q3 are somewhat ambiguous. A subsequent consultation was made with the participants asking about this result, and the conclusion was that the question was not very clear to them, hence the disparity in the responses.

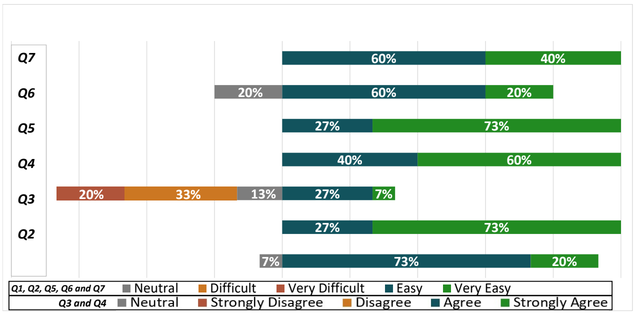


Figure 11. Participant Responses for G2 Questions.

As mentioned above, the objectives of our evaluation were to demonstrate that L-PRISM is efficient, productive, and easy to use. The findings, supported by quantitative and qualitative data, highlight the strengths of L-PRISM in facilitating the creation and management of *multimedia SFCs*. Furthermore, based on the results obtained, we can conclude that goals *G1* and *G2* were achieved.

6.4.3 Derived Metrics and Structural Complexity

Beyond execution time and previously reported subjective perception, we analyzed the structural elements of each task (*T1, T2, T3,* and *T4*) to obtain additional metrics related to developer effort and modeling productivity when using L-PRISM. Table 25 details the number of *multimedia VNFs*, *Virtual Devices (VDs)*, and *Virtual Links (VLs)* defined in each task, as well as the corresponding lines of code and the average time required to complete each task using L-PRISM.

Structural complexity in our context refers to the total number of elements that compose a model instance (i.e., a *multimedia SFC*) namely *multimedia VNFs*, *VDs*, and *VLs*. This type of measure has been used in various modeling approaches to estimate the comprehension and maintenance effort of system models [Briand *et al.*, 1996].

To understand how this complexity affects modeling effort, we derived two complementary metrics:

- Lines of Code per Component (LoC/Comp): This metric captures the verbosity of the language in relation to the total number of structural elements and is defined as:

$$\text{LoC/Comp} = \frac{\text{LoC}}{\#\text{mVNFs} + \#\text{VDs} + \#\text{VLs}}$$

- Time per Component (T/Comp): This metric evaluates modeling efficiency based on the time spent per element and is defined as (in seconds (s)):

$$\text{T/Comp} = \frac{\text{T} \times 60}{\#\text{mVNFs} + \#\text{VDs} + \#\text{VLs}}$$

Where:

- **LoC**: Lines of code written to define the multimedia SFC in L-PRISM;
- **T**: Average time (in minutes) required by participants to complete the task;
- **#mVNFs**: Number of *multimedia VNFs* in the task;
- **#VDs**: Number of *Virtual Devices* involved;
- **#VLs**: Number of *Virtual Links* connecting the components.

Table 25. Derived metrics per task.

Task	#mVNFs	#VDs	#VLs	LoC	T(min)	LoC/Comp	T/Comp(s)
T1	1	1	2	46	4.86	11.5	72.9
T2	2	1	4	79	5.21	11.3	44.7
T3	3	2	6	115	6.64	10.4	36.2
T4	3	2	6	115	1.14	10.4	6.2

This quantitative analysis yields the following key insights:

- The LoC/Comp metric decreases slightly as the number of elements grows across tasks, indicating that L-PRISM provides a consistent level of expressiveness without increasing redundancy as complexity grows. This suggests that the language is concise and well-suited for modeling scenarios with increasing structural demands;
- The stability in LoC/Comp values confirms that the model size in lines of code grows proportionally with the number of structural elements, which is desirable in well-designed modeling languages [Briand *et al.*, 1996];
- Regarding Time/Comp, we observe a progressive reduction even as the total number of components increases from Task T1 (4) to T2 (7) and T3 (11). This trend suggests improved modeling efficiency, possibly due to the growing familiarity of the participants with L-PRISM, and potentially also the reuse of previously defined elements. Notably, even if no such reuse occurred between tasks, the decrease in Time/Comp highlights the effectiveness of the language in supporting developers as the structural complexity of a *multimedia SFC* increases.

These derived metrics offer an additional quantitative perspective on the scalability, expressiveness, flexibility, and developer efficiency enabled by L-PRISM, reinforcing the findings presented earlier.

6.5 Limitations

Although our evaluation results were very positive, there are some limitations of our study that we depict below:

- **Participants characterization:** The test group consisted of 15 participants, most of whom had no prior experience in multimedia system implementation. Although all participants received the same amount of training time, their different learning curves may have influenced their assimilation of the DSL. This variation in learning speed, combined with their lack of prior domain experience, and the limited number of participants may have affected the overall results, limiting the generalizability of our conclusions;
- **Metamodel evaluation:** During the experiments, we focused on evaluating the use of L-PRISM within the V-PRISM architecture. Although the M-PRISM metamodel is designed to enable the creation of other DSLs, we did not implement or evaluate its application in other languages. Therefore, its scope and adaptability have not yet been fully analyzed with our experiment.

7 L-PRISM Analysis Using DSL Evaluation Criteria

In this section, we present a qualitative analysis of L-PRISM. Based on the literature, [Brambilla *et al.*, 2017], [Poltronieri *et al.*, 2018] and [Poltronieri *et al.*, 2024] we identified two relevant criteria commonly used to evaluate DSLs qualitatively. The selected criteria are

- **Flexibility**: ability of the language to adapt to new requirements or extensions in its domain without requiring major changes to its core structure [Brambilla *et al.*, 2017].
- **Representativeness**: the extent to which the language can accurately represent the key concepts and scenarios of the real-world DSL domain it is intended to model [Poltronieri *et al.*, 2018].

7.1 Flexibility

In this section, we present scenarios that are not currently supported by L-PRISM, but could be addressed through extensions to the language. These examples illustrate the L-PRISM flexibility and how the metamodel could evolve to meet new requirements. Some possible extensions include support for (i) energy-aware *multimedia SFC* composition, (ii) billing and quota enforcement, and (iii) dynamic adaptation based on security policies.

- **Energy-Aware Service Composition:** L-PRISM currently does not model energy consumption or battery levels of edge and multimedia devices. In scenarios like drone-based surveillance, energy efficiency is a key requirement. By extending the metamodel to include energy constraints, it would be possible to create modules that prioritize VNFs or paths based on energy availability.
- **Billing Enforcement:** commercial deployments may require to adapt a *multimedia SFC* when users reach a subscription limit. In this scenario, replacing premium *multimedia SFCs* with basic ones must be done. L-PRISM does not have economic constraints, but a billing and quotas modules could enable such dynamic service reconfiguration.
- **Security Adaptation**: In sensitive domains like telemedicine, services may need to reconfigure based on detected threats or trust levels. For example, adding encryption or anonymization VNFs in certain paths. While L-PRISM does not support conditional security policies yet, it could be extended with a security policy module to enforce security in *multimedia SFC*.

To support these new scenarios, additional capability types must be introduced in the language specification. Once specified, corresponding attributes of these types should be added to the *chainModel* entity. Table 26 provides an example, specifically illustrating how the billing enforcement scenario can be incorporated into L-PRISM. It is important to note that the existing modules and attributes remain unchanged.

Table 26. New billing capability type definition.

Name	Description
planName* **Type:** *String*	The billing plan name.
bwLimit* **Type:** *float*	The max bandwidth allowed to be consumed in the *multimedia SFC*.
bwConsumed* **Type:** *float*	The max bandwidth that clients associated with this plan can consume.
bwReduction* **Type:** *float*	Specifies the percentage reduction of bwLimit once the bwConsumed threshold is reached.

The presented scenarios could be addressed through extensions of L-PRISM without major changes in the language core components. This demonstrates the ability of the language to adapt to new requirements while maintaining its original structures. Such examples highlight the flexibility of our approach and its potential for use in novel scenarios.

7.2 Representativeness

Representativeness refers to how well a modeling language captures the key concepts and scenarios of its target domain. In the case of L-PRISM, the language enables modeling of multiple aspects of *multimedia SFC*, such as multimedia stream source, VNF required resources, and VNF chaining.

In this section, we present a possible scenario that can be described using L-PRISM.

In a smart city, when a traffic accident is detected by surveillance cameras, a multimedia emergency response system is triggered. It automatically processes video streams from cameras at the accident location. These streams are used to extract relevant frames for object detection, such as car license plates, vehicle types, anonymizes pedestrian faces, and sends the result stream in real time to traffic control and emergency responders.

The multimedia stream data sources are traffic cameras, microphones near the location, and body cams from the rescue team. These data sources are modeled as virtual devices. L-PRISM can describe a multimedia system that deals with this heterogeneity of multimedia data sources.

Multiple *multimedia SFC* can be created based on the data sources available. In this scenario, a relevant SFC can be created based on the audio from the microphone, which can be processed to convert it to text and subsequently extract relevant events. Another SFC can be created combining the video from multiple cameras to create a multi-angle view. All these SFCs can be described using L-PRISM.

Finally, the edge-cloud environment can be mobilized to execute the VNFs of the SFCs. Edge nodes near the incident can execute *Object Detector* and *Face Anonymizer* VNFs to minimize latency. Cloud services can handle VNFs that execute the *Event Logger* VNF and long-term storage data. Figure 12 illustrates the physical multimedia sensors involved in the presented scenario. Devices (A) and (B) are cameras that provide video streams, while device (C) is a microphone that captures audio from the accident location.

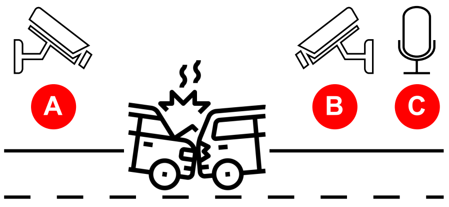


Figure 12. Multimedia sensors monitoring a car accident in a smart city.

Source Code 2 presents a *multimedia SFC* described in L-PRISM, representing a portion of a multimedia system designed to perform traffic accident monitoring in a smart city. This example shows that L-PRISM is representative, as it supports the implementation of a real-world scenario using its native features. We highlight the main components and their interactions. This shows L-PRISM's ability to capture key domain elements effectively to create relevant *multimedia SFCs*.

8 Conclusion

This work presented a metamodel designed to describe multimedia applications named M-PRISM. Based on M-PRISM, we developed a DSL named L-PRISM, specifically tailored for the specification of *multimedia SFCs*. We also implemented a PoC named ALFA 2.0 with

Source Code 2: A multimedia SFC for traffic accident monitor in smart city in L-PRISM.

```
chainModel:
    chainName: Traffic Accident Monitor
    chainDescription: Enable the monitor of
        \traffic accidents in a smart city.
    devices:
    - deviceName: RoadCam1
      deviceId: cam1
    - deviceName: RoadCam2
      deviceId: cam2
    - deviceName: RoadMic1
      deviceId: mic1
    vmss:
    -
     vmsName: AudioVideoCombiner
     vmsDescription: Combine two or more video /
         \audio streams
    -
     vmsName: FaceBlur
     vmsDescription: Apply a blur filter to
         \faces in a video stream
    virtuallinks:
    - #virtualLink 1
        source:
          sourceName: mic1
          sourceType: device
          outputType: video
        destination:
          destinationName: AudioVideoCombiner
          port: 10000
          inputType: audio
    - #virtualLink 2
        source:
          sourceName: cam1
          sourceType: device
          outputType: video
        destination:
          destinationName: AudioVideoCombiner
          port: 10001
          inputType: video
    - #virtualLink 3
        source:
          sourceName: cam2
          sourceType: device
          outputType: video
        destination:
          destinationName: AudioVideoCombiner
          port: 10002
          inputType: video
    - #virtualLink 4
        source:
          sourceName: AudioVideoCombiner
          sourceType: vms
          outputType: video
        destination:
          destinationName: FaceBlur
          port: 5000
          inputType: video
    - #virtualLink 5
        source:
          sourceName: FaceBlur
          sourceType: vms
          outputType: video
        destination:
          destinationName: DashBoardInterface
          address: 192.168.0.156
          port: 10101
          inputType: video
```

the ability to execute the SFCs described with L-PRISM. The source code is available on our GitHub repository, where the installation guide and all the necessary resources are provided to install, run, and test our work.

L-PRISM provides a robust DSL for describing multimedia applications in various scenarios such as virtual and augmented reality, live streaming, and surveillance systems. The ALFA 2.0 implementation serves as a powerful platform for deploying these applications as defined using L-PRISM. The practical applicability of L-PRISM, however, depends on the available *multimedia VNFs*, which are the building blocks of *multimedia SFCs*, responsible for processing multimedia streams.

We evaluated L-PRISM, and the results confirmed that our proposal language is efficient, productive, and easy to use. An observation we made was that the learning curve was shorter using our Web Portal to build the *multimedia SFC*, as it focused on learning to use a specific application. In contrast, the learning curve for L-PRISM varied according to the user's prior experience with programming languages; users with greater familiarity adapted more rapidly, and therefore completed the tasks faster.

The adoption of L-PRISM can be enhanced through its integration with *multimedia SFC* components described in other languages, such as TOSCA-NFV. This integration refers to the translation of SFC elements from these languages into L-PRISM, allowing for a unified modeling approach. To support runtime execution across heterogeneous environments, additional interoperability mechanisms, such as those proposed by [Huff *et al.*, 2019], which use VPNs to link components deployed in different domains, may be employed to route the multimedia streams.

In future work, we plan to develop a graphical tool to manage *multimedia SFCs* described using L-PRISM, providing an intuitive user interface to visualize and modify services. Furthermore, our objective is to improve the integration of resource allocation and scaling algorithms within the ALFA 2.0 platform. In terms of infrastructure, we intend to transition our deployment environment to a cloud-native approach, ensuring adaptability to modern infrastructure needs. We also plan to develop new languages based on the proposed metamodel to validate its adaptability in describing different types of multimedia applications. Furthermore, we will evaluate the integration of environments that execute SFCs with L-PRISM and those that utilize other description languages, such as TOSCA. We also intend to adopt a decentralized approach for an extended implementation of our PoC.

Declarations

Acknowledgements

The authors would like to express their gratitude to MídiaCom Lab for providing the necessary infrastructure to conduct the experiments presented in this paper.

Funding

This work was supported by CAPES, CNPq, MCTI, FAPERJ and INCT-ICoNIoT. Flavia C. Delicato and Débora Muchaluat-Saade are CNPq and FAPERJ Fellows.

Authors' Contributions

FQ, AB, DMS and FD contributed to the conception of this study. FQ performed the experiments. FQ and AB conducted the software development; FQ is the main contributor and writer of this manuscript. AB, DMS and FD revised it. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

- The implementation source code is available at: `https://github.com/fventuraq/alfa`;
- The metamodel proposed in this work (M-PRISM) is available at: `https://fventuraq.github.io/metamodel`;
- The user manual for the proposed DSL (L-PRISM) is available at: `https://fventuraq.github.io/lprism`;
- The data collected during the interviews is available from the authors upon request via email.

References

011, E. G. N.-I. (2023). Network functions virtualisation (nfv) release 4; management and orchestration; vnf descriptor and packaging specification. Available at: `https://docbox.etsi.org/ISG/NFV/open/Publications_pdf/Specs-Reports/NFV-IFA%20011v4.5.1%20-%20GS%20-%20VNF%20Packaging%20Spec.pdf`.

014, E. G. N.-I. (2021). Network functions virtualisation (nfv) release 4; management and orchestration; network service templates specification. Available at:`https://docbox.etsi.org/ISG/NFV/open/Publications_pdf/Specs-Reports/NFV-IFA%20014v4.2.1%20-%20GS%20-%20Network%20Service%20Templates%20Spec.pdf`.

Alam, I., Sharif, K., Li, F., Latif, Z., Karim, M. M., Biswas, S., Nour, B., and Wang, Y. (2021). A Survey of Network Virtualization Techniques for Internet of Things Using SDN and NFV. *ACM Computing Surveys*, 53(2):1–40. DOI: 10.1145/3379444.

Allouche, M., Mitrea, M., Moreaux, A., and Kim, S.-K. (2021). Automatic smart contract generation for internet of media things. *ICT Express*, 7(3):274–277. DOI: 10.1016/j.icte.2021.08.009.

Alvarez, F., Breitgand, D., Griffin, D., Andriani, P., Rizou, S., Zioulis, N., Moscatelli, F., Serrano, J., Keltsch, M., Trakadas, P., Phan, T. K., Weit, A., Acar, U., Prieto, O., Iadanza, F., Carrozzo, G., Koumaras, H., Zarpalas, D., and Jimenez, D. (2019). An edge-to-cloud virtualized multimedia service platform for 5g networks.

IEEE Transactions on Broadcasting, 65(2):369–380. DOI: 10.1109/TBC.2019.2901400.

Battisti, A., Muchaluat-Saade, D. C., and Delicato, F. C. (2020a). Uma proposta de arquitetura para virtualização de sensores multimídia na borda da rede. In *Anais do XXV Workshop de Gerência e Operação de Redes e Serviços (WGRS 2020)*, WGRS 2020, page 235–248. Sociedade Brasileira de Computação - SBC. DOI: 10.5753/wgrs.2020.12464.

Battisti, A. L. E., Macedo, E. L. C., Josué, M. I. P., Barbalho, H., Delicato, F. C., Muchaluat-Saade, D. C., Pires, P. F., Mattos, D. P. d., and Oliveira, A. C. B. d. (2022). A novel strategy for vnf placement in edge computing environments. *Future Internet*, 14(12):361. DOI: 10.3390/fi14120361.

Battisti, A. L. E., Muchaluat-Saade, D. C., and Delicato, F. C. (2020b). *V-PRISM: An Edge-based Architecture to Virtualize Multimedia Sensors in the Internet of Media Things*. Master dissertation, Universidade Federal Fluminense. Available at:`https://sucupira-legado.capes.gov.br/sucupira/public/consultas/coleta/trabalhoConclusao/viewTrabalhoConclusao.jsf?popup=true&id_trabalho=9333087`.

Battisti, A. L. E., Muchaluat-Saade, D. C., and Delicato, F. C. (2020c). V-prism: An edge-based iot architecture to virtualize multimedia sensors. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6. DOI: 10.1109/WF-IoT48130.2020.9221199.

Battisti, A. L. E., Muchaluat-Saade, D. C., and Delicato, F. C. (2021). Enabling internet of media things with edge-based virtual multimedia sensors. *IEEE Access*, 9:59255–59269. DOI: 10.1109/ACCESS.2021.3073240.

Bhamare, D., Jain, R., Samaka, M., and Erbad, A. (2016). A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155. DOI: 10.1016/j.jnca.2016.09.001.

Björklund, M. (2010). YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). (6020). DOI: 10.17487/RFC6020.

Blackwell, A. F. and Green, T. R. (2000). A cognitive dimensions questionnaire optimised for users. In *PPIG*, volume 13. Citeseer. Available at:`https://www.ppig.org/files/2000-PPIG-12th-blackwell.pdf`.

Borum, H. S. and Seidl, C. (2022). Survey of established practices in the life cycle of domain-specific languages. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, MODELS '22, page 266–277, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3550355.3552413.

Brambilla, M., Cabot, J., and Wimmer, M. (2017). *Model-driven software engineering in practice*. Morgan & Claypool Publishers. DOI: 10.1007/978-3-031-02549-5.

Briand, L., Morasca, S., and Basili, V. (1996). Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–86. DOI: 10.1109/32.481535.

Castillo-Lema, J., Venâncio Neto, A., de Oliveira, F., and Takeo Kofuji, S. (2019). Mininet-nfv: Evolving mininet with oasis tosca nvf profiles towards reproducible nfv prototyping. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 506–512. DOI: 10.1109/NETSOFT.2019.8806686.

Coêlho, R. W., Silva, R. A., Martimiano, L. A. F., and Leonardo, E. J. (2024). Iot and 5g networks: A discussion of sdn, nfv and information security. *Journal of the Brazilian Computer Society*, 30(1):212–227. DOI: 10.5753/jbcs.2024.3021.

Das, H., Naik, B., Pati, B., and Panigrahi, C. R. (2014). A survey on virtual sensor networks framework. *International Journal of Grid Distribution Computing*, 7(5):121–130. DOI: http://dx.doi.org/10.14257/ijgdc.2014.7.5.11.

Di Mauro, M., Galatro, G., Longo, M., Postiglione, F., and Tambasco, M. (2021a). Comparative performability assessment of sfcs: The case of containerized ip multimedia subsystem. *IEEE Transactions on Network and Service Management*, 18(1):258–272. DOI: 10.1109/TNSM.2020.3044232.

Di Mauro, M., Galatro, G., Longo, M., Postiglione, F., and Tambasco, M. (2021b). Hasfc: A mano-compliant framework for availability management of service chains. *IEEE Communications Magazine*, 59(6):52–58. DOI: 10.1109/MCOM.001.2000939.

ETSI, G. N.-S. . (2022). Network functions virtualisation (nfv) release 4; protocols and data models; nfv descriptors based on tosca specification.

Farahani, R., Bentaleb, A., Timmerer, C., Shojafar, M., Prodan, R., and Hellwagner, H. (2023). Sarena: Sfc-enabled architecture for adaptive video streaming applications. In *ICC 2023 - IEEE International Conference on Communications*, pages 864–870. DOI: 10.1109/ICC45041.2023.10279262.

Garcia, B., Lopez-Fernandez, L., Gallego, M., and Gortazar, F. (2017). Kurento: The swiss army knife of webrtc media servers. *IEEE Communications Standards Magazine*, 1(2):44–51. DOI: 10.1109/MCOMSTD.2017.1700006.

Ghorab, A., Kusedghi, A., Nourian, M. A., and Akbari, A. (2020). Joint vnf load balancing and service auto-scaling in nfv with multimedia case study. In *2020 25th International Computer Conference, Computer Society of Iran (CSICC)*, pages 1–7. DOI: 10.1109/CSICC49403.2020.9050122.

Huff, A., Venâncio, G., and Duarte Jr., E. P. (2019). Multi-sfc: Orquestração de sfcs distribuídas sobre múltiplas nuvens em múltiplos domínios com múltiplas plataformas nfv. In *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2019)*, SBRC 2019, page 777–790. Sociedade Brasileira de Computação. DOI: 10.5753/sbrc.2019.7402.

Imagane, K., Kanai, K., Katto, J., Tsuda, T., and Nakazato, H. (2018). Performance evaluations of multimedia service function chaining in edge clouds. In *CCNC*, pages 1–4. DOI: 10.1109/CCNC.2018.8319249.

Kalan, R. and Dulger (2024). A survey on qoe management schemes for http adaptive video streaming: Challenges, solutions, and opportunities. *IEEE Access*, 12:170803–

170839. DOI: 10.1109/ACCESS.2024.3491613.

Khan, A. A., Laghari, A. A., Shaikh, A. A., Shaikh, Z. A., and Jumani, A. K. (2022). Innovation in multimedia using iot systems. *Multimedia computing systems and virtual reality*, pages 171–187. DOI: 10.1201/9781003196686-8.

Koziolek, H. (2008). *Goal, Question, Metric*, pages 39–42. Springer Berlin Heidelberg, Berlin, Heidelberg. DOI: 10.1007/978-3-540-68947-8_6.

Macedo, E. L. C., Battisti, A. L. E., Vieira, J. L., Noce, J., Pires, P. F., Muchaluat-Saade, D. C., Oliveira, A. C. B., and Delicato, F. C. (2023). Distributed auction-based sfc placement in a multi-domain 5g environment. *SN Computer Science*, 5(1). DOI: 10.1007/s42979-023-02291-1.

Maia, A., Boutouchent, A., Kardjadja, Y., Gherari, M., Soyak, E. G., Saqib, M., Boussekar, K., Cilbir, I., Habibi, S., Ali, S. O., Ajib, W., Elbiaze, H., Erçetin, O., Ghamri-Doudane, Y., and Glitho, R. (2024). A survey on integrated computing, caching, and communication in the cloud-to-edge continuum. *Computer Communications*, 219:128–152. DOI: 10.1016/j.comcom.2024.03.005.

Manias, D. M., Shaer, I., Naoum-Sawaya, J., and Shami, A. (2024). Robust and reliable sfc placement in resource-constrained multi-tenant mec-enabled networks. *IEEE Transactions on Network and Service Management*, 21(1):187–199. DOI: 10.1109/TNSM.2023.3293027.

Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. (2017). A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Comm. Surveys and Tutorials*, 19(4):2322–2358. DOI: 10.1109/COMST.2017.2745201.

Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., De Turck, F., and Boutaba, R. (2016). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*, 18(1):236–262. DOI: 10.1109/COMST.2015.2477041.

Nauman, A., Qadri, Y. A., Amjad, M., Zikria, Y. B., Afzal, M. K., and Kim, S. W. (2020). Multimedia internet of things: A comprehensive survey. *IEEE Access*, 8:8202–8250. DOI: 10.1109/ACCESS.2020.2964280.

Negm, E., Makady, S., and Salah, A. (2019). Survey on domain specific languages implementation aspects. *International Journal of Advanced Computer Science and Applications*, 10(11). DOI: 10.14569/IJACSA.2019.0101183.

Neto, M. C., Andrade, S. S., and Novais, R. L. (2017). Cross-platform multimedia application development: for mobile, web, embedded and iot with qt/qml. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, pages 23–26. DOI: 10.1145/3126858.3131627.

Poltronieri, I., Zorzo, A. F., Bernardino, M., and de Borba Campos, M. (2018). Usa-dsl: usability evaluation framework for domain-specific languages. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, page 2013–2021, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3167132.3167348.

Poltronieri, I., Zorzo, A. F., Bernardino, M., and OliveiraJr, E. (2024). Usa-dsl: a process for usability evaluation of domain-specific languages. *Journal of Universal Computer Science*, 30(8):1023. DOI: 10.3897/jucs.103264.

Quico, F. J. V., Battisti, A. L. E., Muchaluat-Saade, D., and Delicato, F. C. (2024). A domain-specific language for multimedia service function chains based on virtualization of sensors. In *Proceedings of the 30th Brazilian Symposium on Multimedia and the Web (WebMedia 2024)*, WebMedia 2024, page 11–19. Sociedade Brasileira de Computação - SBC. DOI: 10.5753/webmedia.2024.243129.

Quico, F. J. V., Muchaluat-Saade, D. C., and Delicato, F. C. (2023). *L-PRISM: A Specification Language for Multimedia Service Chains Based on Virtualization of Sensors*. Master dissertation, Universidade Federal Fluminense. Available at:`https://sucupira-legado.capes.gov.br/sucupira/public/consultas/coleta/trabalhoConclusao/viewTrabalhoConclusao.jsf?popup=true&id_trabalho=13718352`.

Rodrigues, D. O., De Souza, A. M., Braun, T., Maia, G., Loureiro, A. A. F., and Villas, L. A. (2023). Service provisioning in edge-cloud continuum: Emerging applications for mobile devices. *Journal of Internet Services and Applications*, 14(1):47–83. DOI: 10.5753/jisa.2023.2913.

Rodrigues, I. P., Zorzo, A. F., and da Silveira, M. B. (2019). Uma proposta de processo de avaliação de usabilidade para dsls. *ERES-Escolar Regional de Engenharia de Software-Trilha Pós-graduação, 2019, Brasil.*. DOI: 10.5753/eres.2021.18455.

Roges, L. and Ferreto, T. (2025). Multi-objective dynamic registry provisioning in edge computing infrastructures. *Concurrency and Computation: Practice and Experience*, 37(4-5):e70006. DOI: 10.1002/cpe.70006.

Santos, G. L., Bezerra, D. d. F., Rocha, □. d. S., Ferreira, L., Moreira, A. L. C., Gonçalves, G. E., Marquezini, M. V., Recse, a., Mehta, A., Kelner, J., Sadok, D., and Endo, P. T. (2022a). Service Function Chain Placement in Distributed Scenarios: A Systematic Review. *Journal of Network and Systems Management*, 30(1):1–39. DOI: 10.1007/s10922-021-09626-4.

Santos, H., Rosario, D., Cerqueira, E., and Braun, T. (2022b). Multi-criteria service function chaining orchestration for multi-user virtual reality services. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 6360–6365. DOI: 10.1109/GLOBECOM48099.2022.10000827.

Schönwälder, J., Björklund, M., and Shafer, P. (2010). Network configuration management using netconf and yang. *IEEE Communications Magazine*, 48(9):166–173. DOI: 10.1109/MCOM.2010.5560601.

Sisinni, E., Flammini, A., Gaffurini, M., Pasetti, M., Rinaldi, S., and Ferrari, P. (2024). Lorawan end device disaggregation and decomposition by means of lightweight virtualization. *Internet of Things*, 25:101033. DOI: 10.1016/j.iot.2023.101033.

Surendran, P. (2012). Technology Acceptance Model: A Survey of Literature. *International Journal of*

Business and Social Research, 2(4):175–178. Available at:`https://ideas.repec.org/a/mir/mirbus/v2y2012i4p175-178.html`.

TOSCA, O. (2017). Tosca simple profile for network functions virtualization (nfv) version 1.0, committee specification draft 04. Available at:`https://docs.oasis-open.org/tosca/tosca-nfv/v1.0/csd04/tosca-nfv-v1.0-csd04.html`.

YAML (2021). YAML Ain't Markup Language (YAML™) Version 1.2. Available at:`https://yaml.org/spec/1.2/spec.html` Accessed: Jun 29, 2024.

Yi, B., Wang, X., Li, K., Das, S. k., and Huang, M. (2018). A comprehensive survey of Network Function Virtualization. *Computer Networks*, 133:212–262. DOI: 10.1016/j.comnet.2018.01.021.

Zerifi, M., Ezzouhairi, A., Boulaalam, A., and Baghrous, M. (2025). Iot challenges and issues: A comprehensive review of software defined networking and network function virtualization solutions. *International Journal of Interactive Mobile Technologies (iJIM)*, 19(03):pp. 209–226. DOI: 10.3991/ijim.v19i03.49791.