VeriFog: A Generic Model-based Approach for Verifying Fog Systems at Design Time

Hiba Awad, Abdelghani Alidra, Hugo Bruneliere, Thomas Ledoux, Etienne Leclerq, Jonathan Rivalan

VeriFog: A Generic Model-based Approach for Verifying Fog Systems at Design Time

Hiba Awad
IMT Atlantique, LS2N (UMR CNRS 6004), Inria Rennes, Smile
Asnières-sur-Seine and Nantes
France
hiba.awad@smile.fr

Abdelghani Alidra
IMT Atlantique, LS2N (UMR CNRS 6004), Inria Rennes
Nantes, France
abdelghani.alidra@imt-atlantique.fr

Hugo Bruneliere
IMT Atlantique, LS2N (UMR CNRS 6004)
Nantes, France
hugo.bruneliere@imt-atlantique.fr

Thomas Ledoux
IMT Atlantique, LS2N (UMR CNRS 6004), Inria Rennes
Nantes, France
thomas.ledoux@imt-atlantique.fr

Etienne Leclerq
Smile
Asnières-sur-Seine, France
etienne.leclerq@smile.fr

Jonathan Rivalan
Smile
Asnières-sur-Seine, France
jonathan.rivalan@smile.fr

ABSTRACT

Fog Computing is a paradigm aiming to decentralize the Cloud by geographically distributing away computation, storage, network resources and related services. It provides several benefits such as reducing the number of bottlenecks, limiting unwanted data movements, etc. However, managing the size, complexity and heterogeneity of Fog systems to be designed, developed, tested, deployed, and maintained, is challenging and can quickly become costly. According to best practices in software engineering, verification tasks could be performed on system design prior to its actual implementation and deployment. Thus, we propose a generic model-based approach for verifying Fog systems at design time. Named VeriFog, this approach is notably based on a customizable Fog Modeling Language (FML). We experimented with our approach in practice by modeling three use cases, from three different application domains, and by considering three main types of non-functional properties to be verified. In direct collaboration with our industrial partner Smile, the approach and underlying language presented in this paper are necessary steps towards a more global model-based support for the complete life cycle of Fog systems.

CCS CONCEPTS

• **Software and its engineering → Model-driven software engineering**; **Software verification and validation**; • **Computer systems organization → Cloud computing**.

KEYWORDS

Model-based Engineering, Modeling Language, Fog Computing, Verification, Non-functional Properties, Design Time.

SAC '24, April 8–12, 2024, Avila, Spain

ACM Reference Format:
Hiba Awad, Abdelghani Alidra, Hugo Bruneliere, Thomas Ledoux, Etienne Leclerq, and Jonathan Rivalan. 2024. VeriFog: A Generic Model-based Approach for Verifying Fog Systems at Design Time. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3605098.3635973

1 INTRODUCTION

Fog Computing [23, 30] is a recent paradigm aiming to decentralize the Cloud by geographically distributing away computation, storage and network resources as well as related services. Instead of a centralized Cloud system, data centers of various sizes in the core network, and smaller data-centers or devices at the edge of the network, can be used collaboratively to form a single large-scale geo-distributed system. Thus, Fog Computing is at the crossroads of complementary areas of distributed systems: Cloud Computing [5] of course, but also Edge Computing [25] and IoT [18].

Over the last years, Fog systems have been quite intensively studied in the research community, though their actual dissemination in industry remains relatively limited [13]. As discussed with our industrial partner Smile, several reasons can be given to explain this situation. A first one is the size, complexity and heterogeneity of the systems to be designed, developed, tested, deployed and then maintained in such a Fog Computing context. Indeed, a Fog system can combine large-scale Cloud resources with a possibly huge number of IoT devices at the Edge of the network. All these varied Fog resources have to be organized into several, sometimes many, inter-connected areas. As a result, such a Fog system can quickly become very challenging and costly to manage efficiently [3]. An important goal is notably to ensure an appropriate Quality of Service (QoS) [27]. This may concern security [21], energy [20], or resource usages [1] aspects (among others).

Best practices in software engineering already showed that a common way to limit development and management costs is to verify the system design prior to its actual implementation and deployment. However, up to our current knowledge, the verification approaches that have been applied so far in the context of Fog systems mostly concern already implemented and/or deployed systems [16]. Moreover, by looking at a detailed study of the state

of the art [4], we observed that system modeling is currently rather limited in Fog Computing and mostly concerns the simulation of only parts of the Fog systems. Thus, to go a step further, we introduce the VeriFog model-based approach for the verification of Fog systems at design time. To this end, we notably propose a generic and customizable Fog Modeling Language (FML) to specify Fog system models that can then be navigated and queried accordingly. To experiment with our approach and language in practice, we tested them for three main types of important non-functional properties (security, energy and performance) that we verified in the context of three different Fog systems we modeled. These Fog systems come from three different use cases illustrating various application domains of Fog Computing. The objective is to demonstrate the practical relevance of VeriFog for verifying several non-functional properties of Fog systems during their design phase. We also want to show the actual usability of FML for modeling different kinds of Fog systems. On the longer term, both VeriFog and FML belong to a wider research initiative intending to capitalize on Fog system models in order to better support the Fog system's overall life cycle.

This paper is organized as follows. Section 2 presents the general background and motivation for our work, resulting in the three research questions we intend to address in this paper. Then, Section 3 introduces the overall VeriFog approach and its targeted verification support. Section 4 describes the FML language as a core component of VeriFog. Then, Section 5 illustrates the practical experiments with our approach and language in the context of three different use cases. Section 6 summarizes the current Eclipse/EMF-based implementation of VeriFog, FML, and the studied use cases. Section 7 discusses the current status, limitation, and lessons learned from the work. Finally, Section 8 explains the related work while Section 9 concludes the paper by opening on future research perspectives.

2 BACKGROUND AND MOTIVATION

As introduced earlier, managing the size, complexity and heterogeneity of Fog systems can rapidly become both challenging and expensive. Providing an answer towards the resolution of this kind of problem is a main objective of the ongoing SeMaFoR project [3], for example. In the context of this collaborative research effort involving both research and industrial partners, we aim at proposing an approach for supporting the self-management of Fog resources. More particularly, we are currently designing and developing a generic, end-to-end, decentralized and collaborative self-management solution to be able to operate different kinds of Fog systems. To this end, we are notably tackling both Fog modeling concerns and their potential impact on various aspects of the Fog system's life cycle. The work presented in this paper is positioned at the intersection of these two dimensions.

On the one hand, Fog Computing remains a relatively recent paradigm. Thus, many aspects related to Fog systems and the general support for their engineering and management are open challenges [13]. In the scientific literature, these aspects have been quite studied from a theoretical and/or research perspective over the past years. Still, for various reasons, it appears that they have yet to be experimented in practice within the industrial context. Among the different phases of the Fog system's life cycle which are worth exploring, the design phase is particularly important. Notably, any verification to be made onto a given system before its implementation and deployment can be highly beneficial in the long run (e.g., to improve the overall QoS of the system [27]). However, existing verification approaches have been mostly applied on Fog systems already implemented and/or deployed [16].

On the other hand, the approach resulting from a project such as SeMaFoR is meant to rely on a language allowing to model different Fog systems, coming from different application domains and for possibly covering different purposes. A deep study of the state of the art in terms of existing modeling languages and related capabilities in a Fog Computing context is already available [4]. From this extensive study, we indicated that the already existing modeling support dedicated to Fog systems, their building and their management, is rather limited at this stage. As a consequence, more efforts are needed in order to complement the available (mostly simulation-based) solutions with more elaborated model-based solutions targeting Fog systems in particular.

Based on this analysis, we considered the following three Research Questions (RQs) in the remainder of the paper:

RQ1. Can we build a (model-based) approach for verifying Fog systems before their implementation or deployment?
RQ2. In such an approach, which language do we need to support the modeling of Fog systems?
RQ3. Can such an approach and language be used for verifying different non-functional properties on different types of Fog systems within different application domains?

3 AN APPROACH FOR DESIGN-TIME VERIFICATION OF FOG SYSTEMS (RQ1)

Figure 1 presents an overview of the VeriFog iterative approach.

(1) The Fog System Architect (FSA) is the main actor. Ideally, she/he is an engineer having the required knowledge and experience in terms of both Fog infrastructures and the targeted application domain(s) (e.g., smart cities, industrial IoT, autonomous vehicles). In practice, it can rather be a collaboration between several Fog system's engineers (e.g., experts on distributed systems) and domain experts (e.g., persons in charge of the Fog system).

(2) Using an appropriate Fog Modeling Language, the FSA starts by modeling the topology of the Fog system(s) she/he wants to design via one or several models/files. This way, she/he identifies and represents the various types of Fog resources that will compose the target Fog system, as well as the various types of possible interconnections between these types of resources. In some cases, the FSA proposes one single topology that can then be instantiated in different configurations of the target Fog system. In other cases, she/he may also propose several topologies to first study the pros and cons of each one of them (before moving to the next step).

(3) Based on the previously defined topology, or the selected one if several have been defined in the previous step, the FSA can specify different Fog system's configurations via several models/files. This way, she/he can represent different possible states of the target Fog systems at various given points in time. These configurations directly refer to the corresponding topology: they contain concrete instantiations of the various types of Fog resources described in the topology, as well as their actual interconnections.

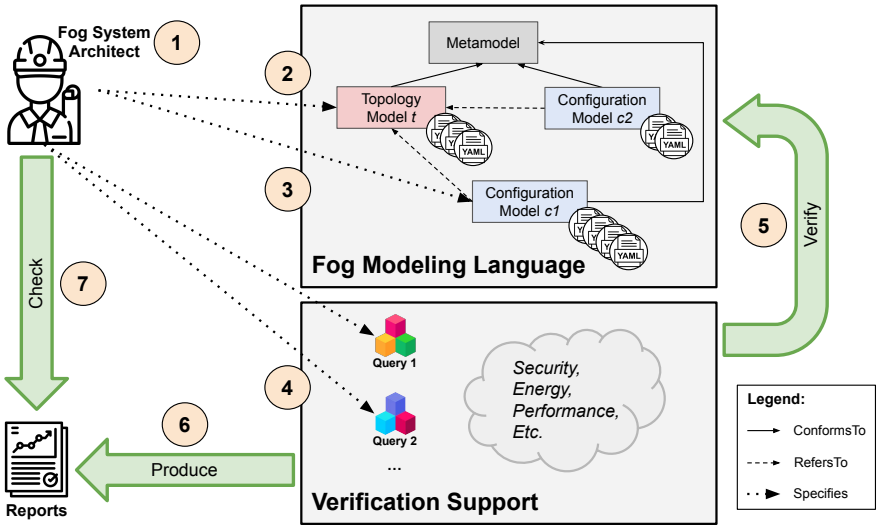

Figure 1: Overview of the VeriFog iterative approach (the circled numbers correspond to numbered paragraphs in the text).

④ Once the required topology and configuration models are properly specified, that all conform to the metamodel of the used Fog Modeling Language, the FSA can identify and express the non-functional properties she/he wants to verify on the target Fog system. As later displayed in Table 1, each property to be verified can be associated to different queries in order to be able to provide a global assessment over the target Fog system. In practice, these queries have to be expressed using a (model) query language (in our current experiments, we used OCL[1]) that is compatible with the Fog Modeling Language we propose (cf. Section 6).

⑤ Once the required queries are properly specified, they can be actually verified onto the set of Fog system's models. Depending on the kind of verification to be made on the Fog system, a given query can be executed over a topology model, a configuration model, or eventually a combination of them. Depending on the type of a given query, the result returned by the execution of this query can take different types: a numerical value, a boolean value or a string textually describing the result. Various examples of non-functional properties and associated queries are presented in Section 5.

⑥ Once all the required queries are executed over the corresponding Fog system's models, a report can be produced to aggregate the individual query execution results. Such a report can take different forms: a simple list of query execution results (as in our current implementation of the approach, cf. Section 6), a more structured and elaborated document automatically generated from the Fog system's models and the results, a tool displaying in a more dynamic way the content of both these models and results, etc.

⑦ Finally, the FSA can consult and check this report. She/he can use it as a valuable input for taking decisions on whether to 1) proceed with the current Fog system's design and eventually start to work on its implementation, or 2) perform another iteration of the VeriFog approach to improve the system's design. In the latter case, the FSA may decide to rework the initial topology model by completing the various types of possible Fog resources. She/he may also decide to first experiment with other configurations before actually modifying the topology, etc.

[1]https://projects.eclipse.org/projects/modeling.mdt.ocl

4 A GENERIC LANGUAGE FOR MODELING FOG SYSTEMS (RQ2)

In the context of the SeMaFoR project [3], we extensively studied the current state of the art in terms of existing languages and related support for modeling Fog systems [4]. However, none of the available solutions appears to come with a generic reusable language that would allow engineers to easily specify, share, maintain and evolve Fog system's models for different kinds of Fog architectures. As a result, and also capitalizing on other existing work on Cloud modeling [2, 10], we designed and developed a generic and customizable Fog Modeling Language (FML). In what follows, we present both its abstract syntax (i.e., its core metamodel) and current concrete syntax (i.e., a textual notation). Its semantics is associated to the language usage (cf. Section 5).

4.1 Abstract Syntax - Metamodel

The main objective of the FML is to allow specifying the various resources composing the Fog system as well as the relationships between them. To this end, the proposed language is able to represent both the different possible *topologies* of Fog systems (i.e., the types of Fog resources) and the corresponding *configurations* based on these topologies (i.e., the instances of these types). Figure 2 shows a partial version of the metamodel of the language.

The main elements of a Fog system are considered as *FogResources*. As the language is meant to be generic and extensible, any *FogResource* can be customized by adding specific metadata via both *Tag* elements (i.e., key/value pairs) and custom *Attribute* elements. Each *Attribute* is characterized by a *AttributeType* targeting a particular type of Fog resource and having a corresponding *UnitOfMeasurement*. The root of the model is the *FogSystem* containing various *FogAreas* (i.e., Fog sub-systems), themselves containing the other

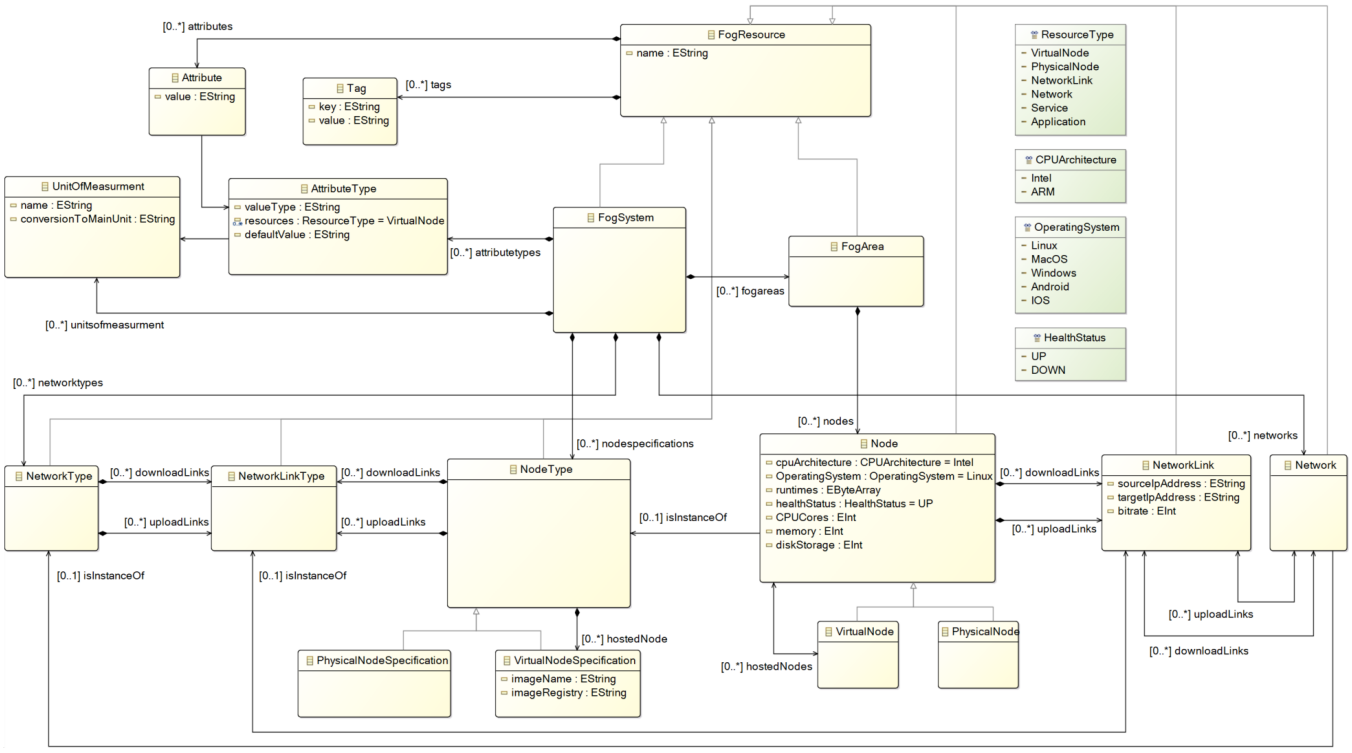

Figure 2: Partial metamodel (i.e., abstract syntax) of the proposed Fog Modeling Language.

kinds of elements. This way, as introduced earlier, both topologies and corresponding configurations can be jointly modeled.

On the topology side, at the *FogSystem*-level, different *NodeType*s can be specified. These are either *PhysicalNodeSpecification*s (i.e., for hardware devices) or *VirtualNodeSpecifications* (i.e., for software components). Naturally, a given *NodeType* can host other *NodeType*s. For example, a virtual node can be hosted on a physical node or eventually on another virtual node. The different *NodeType*s can be interconnected via various *NetworkLinkType*s that materialize either download or upload links depending on the case. In addition, each *NetworkLinkType* is related to a given *NetworkType* that can have different characteristics associated to it (e.g., these characteristics can be expressed via particular *Tag*s or custom *Attribute*s).

On the configuration side, at the *FogArea*-level, the overall structure is similar. However, several *Node*s can be interconnected via various *NetworkLink*s (download or upload) directly related to corresponding *Network*s. Each *Node* is representing a particular instance of a previously specified *NodeType*. In addition, such a *Node* carries different general runtime properties concerning its operating system, CPU, memory, disk space, health status, etc. Similarly, each *NetworkLink* and *Network* are particular instances of a previously specified *NetworkLinkType* and *NetworkType* (respectively).

Finally, it is important to notice that some aspects of the language are not described in the paper for the sake of readability and understandability. Indeed, some simple multiplicities (e.g., 0..1 / 1..1) and associated labels are hidden in Figure 2. For the same reason, the provided support for different kinds of constraints over *NodeType* and *NetworkType* elements is not detailed neither in the paper. Note that this part of the language is directly adapted from the constraint support proposed in previous work on Cloud modeling [2, 10]. More importantly, the language also supports the modeling of both the services and related applications that can be hosted in the Fog system nodes. These are also important elements in the context of highly distributed Fog systems. However, such elements are not described here as not used in the current work where we mostly focus on modeling Fog infrastructures.

4.2 Concrete Syntax - YAML Textual Notation

The technical background of the typical FSA usually makes her quite familiar with solutions for distributed system's infrastructures. In such solutions, textual notations like YAML[2] are frequently encountered. As a consequence, we have opted for a YAML textual syntax to accompany the current version of the proposed Fog Modeling Language. Figure 3 shows an excerpt of this textual notation in the context of one of the practical use cases from Section 5.

The different *FogResource*s composing the *FogSystem* can be modeled thanks to separate YAML files (.yml). These files can be organized, stored and shared as needed by the concerned Fog engineers. For example, the left file in Figure 3 shows a *VirtualNodeSpecification* named "Fognode2", as part of a given topology. Three *Tag*s are associated with this particular *NodeType* to add specific performance-related metadata in this particular case. Then, a "spec" section allows to specify the general value of the different properties (both the base and custom ones). This section also allows to

[2]https://yaml.org/

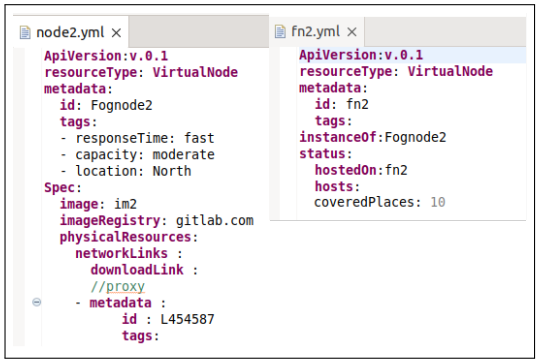

Figure 3: Excerpt of the YAML-like textual notation (i.e., concrete syntax) of the proposed Fog Modeling Language.

declare the corresponding *NetworkLinkTypes* (download and upload ones) and related metadata, etc. Based on this topology specification, the right file in Figure 3 shows a given configuration providing an instantiation of the previously defined "Fognode2" *NodeType*, as a *Node* named "fn2". In a configuration definition, the "spec" section (from the topology specification) is replaced by a "status" section that allows to indicate the runtime value of the different properties, to declare the corresponding *NetworkLink*s, etc.

Note that, to simplify the learning and usage of the proposed notation, we voluntarily decided to have a similar syntax for describing both topology and configuration elements. Only some dedicated sections of the files are different at the two levels, e.g., "spec" vs. "status" as mentioned earlier on the provided example.

5 PRACTICAL APPLICATIONS ON DIFFERENT USE CASES (RQ3)

To evaluate the genericity, and more generally the usability, of our VeriFog approach and underlying Fog Modeling Language, we considered various concrete Fog system's examples coming from different application domains. From the literature, we selected three distinct use cases we found relevant and representative of the heterogeneity of Fog systems. They are summarized in Table 1.

In what follows, for the two first use cases, we provide 1) a textual description of the overall context and concrete application, 2) a model of the corresponding Fog system in our Fog Modeling Language, and 3) examples of queries used to verify non-functional properties which are particularly relevant in highly-distributed Fog systems and that can be verified at design-time. Due to space limitation, we do no detail the third use case. For all use cases, the complete implementation resources can be accessed via the provided open source repository (cf. Section 6).

Note that, in the following sections, the presented models are voluntarily partial. Indeed, for the sake of readability, only the *NodeType* (in red) and *Node* (in blue) elements are displayed. Thus, the root elements (*FogSystem*, *FogArea*) and the network elements (*NetworkType*, *NetworkLinkType*, *Network*, *NetworkLink*) are not shown. Moreover, to keep the diagrams light, the latter are simply materialized by generic dependency relationships. Finally, the custom *Attribute*s are displayed the same way than regular attributes within the corresponding elements.

5.1 Smart Campus: Energy Property

Scenario: The first use case is taken from a published work presenting a recent survey on existing Fog Modeling Languages [4]. In an university campus context, the survey proposes a motivating example of a Fog system that consists in two distributed applications for smart surveillance and smart bell notification, respectively. They are made available as a set of loosely coupled micro-services that can be mutually shared in order to provide the expected capabilities.

Model: The Fog system of the monitored university campus is composed of several *FogAreas* corresponding to the different sections of the campus. We show in Figure 4 the content of two main *FogAreas* for the two sections of our example campus: the Main Department and the Dormitory.

The central elements are "center1" and "center2" data center *VirtualNodes*. They can be connected to other *FogAreas* of the system indirectly via the "gate" gateway *VirtualNode*, itself connected to the "publicCloudProvider" cloud *VirtualNode*. The different nodes composing the system are also connected to these two central data center nodes. In the dormitory, this is the case for the "watch" and "comp" *PhysicalNodes* (of type smart watch and computer, respectively). In the main department, this is the case for the "alarm", "mobile" and "cam" *PhysicalNodes* (of type alarm device, mobile phone and wifi camera, respectively). In this main department, the "cluster" Rapsberry Pi cluster *PhysicalNode* is indirectly connected via the "gate" gateway *VirtualNode*.

At the topology level, we added a "SourceEnergy" custom *Attribute* to model the energy source type associated to each *NodeType* instantiated in the system. Depending on the origin of the energy, it can be renewable (i.e., green), or fossile (as in, oil or petroleum). We also added another "greenEnergy" custom *Attribute* to be able to represent the green energy level associated to each *NodeType*. In case of multiple energy sources, this corresponds to the percentage of green energy over the total energy. For example, the level is considered high if this percentage is higher than 50. At the configuration level, we added a "energy_consumption" custom *Attribute* to be able to describe the amount of the total energy already consumed by each node at a given point in time. We also added another "green_consumption" custom *Attribute* to be able to specify the percentage of the total green energy already consumed by each node at a given point in time.

Queries: In the university campus, energy consumption is an important issue for both ecological and financial reasons. Thus, we need to assess the energy origin and consumption for the different kinds of Fog resources composing the system, as well as for the system as a whole. In what follows, we consider a college campus consisting of a main department and a dormitory.

Query "IsGreen" - To start with, we defined a global query at the topology level in order to assess the "greenness" of the Fog system based on the different types of instantiated nodes. Listing 5 shows the OCL code of this query that returns a boolean value. To summarize, we get the green energy level for each node in the system and we check whether the majority of the nodes have a high green energy level or not.

Query "Remaining Energy" - Then, we defined a query at the configuration level to compute the remaining available energy to be consumed (i.e., a double value) associated to each node type in the

Table 1: Several applications of the VeriFog approach.

Use Case Name	Verified Property	Query	Level	Return Type	Objective
Smart Campus	Energy	IsGreen	Topology	Boolean	Check if the system is green or not, i.e., if the majority of the nodes of this system have a renewable energy resource or not.
		Remaining Energy	Configuration	Double	Study the system energy status by calculating the difference between a consumption threshold and the energy consumption at time T.
		Green Energy Consumption	Configuration	Double	Calculate the percentage of green energy consumption by knowing the green level and consumed energy of each node.
Smart Parking	Performance	Efficiency	Topology	Double	Check if the system is efficient by calculating the parking lot ratio, i.e., the balance between the number of cameras and the covered slots.
		Latency	Configuration	Float	Calculate the system latency, i.e., the delay of analyzing each area of the parking lot and the delay of noise filtering in defected images.
		IsReliable	Configuration	Boolean	Study the system reliability by calculating the number of instances of each camera type with a given detection quality level.
Smart Hospital	Security	IsSecured	Topology	Boolean	Check if the system is "Secured" or not so we can know if most of the nodes in our system are located in a secure environment.
		RiskScore	Topology	Integer	Study the vulnerability of each node, based on the node location and the importance of its protected/stored data.
		Authorized Token Level	Configuration	Sequence	Group the nodes by their authorization level.

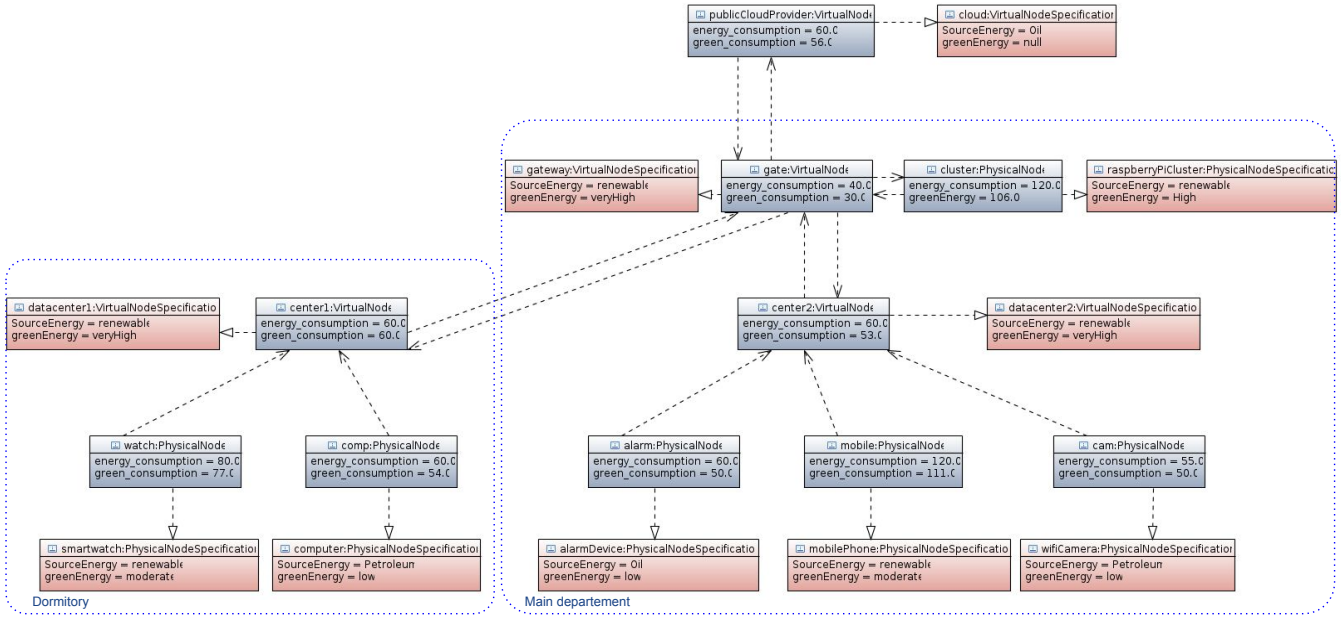

Figure 4: Partial model of the Fog system in the Smart Campus use case.

```
if NodeType.allInstances()->select(f | f.tags->exists(t | t.key =
↪  'greenEnergy' and (t.value = 'low' or t.value = 'null' or
↪  t.value='moderate')))->size() > NodeType.allInstances()->size()/2
↪  then
    'The system is not green'
else
    'The system is Green'
endif
```

Figure 5: OCL code of the IsGreen query in the Smart Campus use case (boolean return value printed in plain readable text).

system. The objective is to calculate the amount of available energy with respect to the threshold imposed by the campus administrators. The calculation is based on 1) the total energy consumption of each node and 2) the threshold of 2000 J. The total energy consumption is the sum of the "energy_consumption" of each node in the system. For example, if this sum is 3000 J, a specific message indicate an over-consumption of 1000 J.

Query "Green Energy Consumption" - Finally, we defined another query at the configuration level to compute the amount of consumed green energy compared to the total consumed energy. The objective is to obtain the percentage of green energy consumed by the system. The calculation is based on 1) the "green_consumption" and 2) the "energy_consumption" of each node.

5.2 Smart Parking: Performance Property

Scenario: The second use case is taken from a published work presenting a solution for drivers to be more efficient when searching for a slot for their cars in a parking lot [6]. Thus, in a parking lot context, the solution describes an example of a Fog system that detect the parked cars in each zone, and indicates available spaces to the drivers via a smart led screen at the entrance.

Model: The Fog system of the monitored parking lot is composed of several *FogAreas*, corresponding to its different sections, that deal with various kinds of nodes (e.g., cameras, micro-controllers). We show in Figure 6 the content of two *FogAreas* but the parking system can be composed of much more *FogAreas*.

The central elements of the shown model are "fogn1" and "fogn2" Fog *VirtualNodes*. These central nodes can be possibly connected to other *FogAreas* of the system, via the "proxyserver" proxy *VirtualNode* and the external "cloudserver" cloud *VirtualNode*. The different devices composing the system (i.e., different types of cameras) are also indireclty connected to these central Fog nodes via "micro1" and "micro2" micro-controller *VirtualNodes*.

At the topology level, we added a "detection_quality" custom *Attribute* to model the detection quality level associated to each camera *NodeType* instantiated in the system. This corresponds to the capability of each camera to always detect any vehicle in the covered zone. For example, if the camera can identify only cars and cannot systematically detect motorcycles, the detection quality will be low. At the configuration level, we added a "coveredPlaces" custom *Attribute* to describe the number of slots covered by each Fog *Node* in the system. This corresponds to the number of slots where cameras can possibly detect the cars. We also added another "image_quality" custom *Attribute* to specify the percentage of clarity for the images detected by a given camera.

Queries: In a city's parking lot context, the overall performance of the system is an important issue in order to be able to provide an efficient service to the drivers. Thus, we need the data to be treated by each camera and Fog node to have a sufficiently high level of quality. For the following queries, we considered a smart parking of a reasonable size with 16 cameras nodes and 2 Fog nodes.

Query "Is Efficient" - To start with, we defined a global query at the topology level to assess the overall efficiency of the parking lot. To this end, we computed the ratio between the number of covered slots and the total number of available cameras. If the ratio is closer to 1, there are possibly too many cameras. If the ratio is closer to 0, there are probably not enough cameras to cover all the slots.

Query "Latency" - Then, we defined a query at the configuration level to compute the time needed for the system to display the available slots on the smart led screen at the entrance of the parking lot. The objective is to evaluate the latency of the overall system. Listing 7 shows the OCL code implementation of this query which returns a real value. To summarize, the calculation is based on 1) the time consumed by the Fog nodes to analyze the images collected from the corresponding 16 cameras (cf. the value from Table 3 in the source paper [6], then encoded in the query) and 2) the time of image enhancement. The time of image enhancement is computed from the "image_quality" of each camera node. If the clarity of the image took by a given camera node is low, an image enhancement is required prior to its analysis.

Query "Reliability" - Finally, we defined another query at the configuration level to assess the reliability of the Fog system based on the different types of instantiated nodes. To summarize, we analyze the image quality of each camera in the system by ensuring that the majority of them have an image quality higher than 50%.

6 IMPLEMENTATION

Based on our own experience and technical expertise, we decided to implement in the Eclipse ecosystem an initial version of the tooling support associated with the proposed Fog Modeling Language, Verification Support, and current use cases. However, both the VeriFog conceptual approach and the FML language specification are technology-independent. As a consequence, they may be redeveloped in other technical environments if required in the future (e.g., by our partner company Smile for industrial purposes).

FML has been implemented by using the Eclipse Modeling Framework (EMF)[3] and related tools. For the abstract syntax, we specified a dedicated metamodel in Ecore. For the concrete syntax, we developed a grammar in Xtext[4] that we connected to this metamodel. This allowed us to produce a corresponding textual editor with basic features (syntax highlighting, code completion, syntax checks, etc.). Thus, we provide an editing environment for Fog System Architects to write their Fog system models using several YAML files. As in the VeriFog approach, they can also get their models as full EMF-compatible models, and eventually serialize them as XMI files for further sharing or usage with other tools.

For the Verification Support, we implemented all the presented queries in two different languages: 1) Java to show the possible use of FML in combination with a general-purpose programming

[3]https://www.eclipse.org/modeling/emf/
[4]https://www.eclipse.org/Xtext/

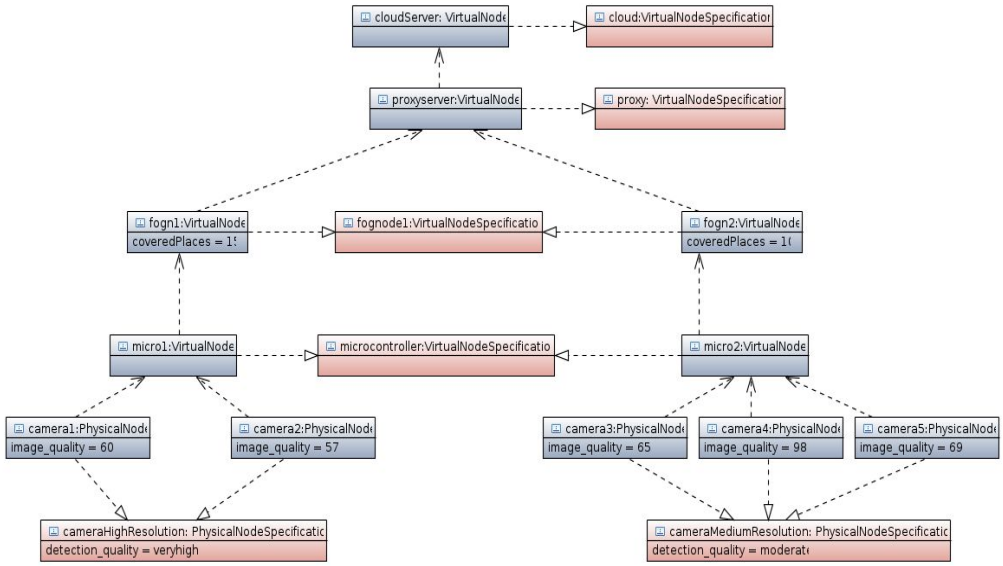

Figure 6: Partial model of the Fog system in the Smart Parking use case.

```
let totalEnhancement: Real =
  NodeType.allInstances()
    ->select(f | f.tags->exists(t | t.key = 'detection_quality' and
    ↪  (t.value = 'low' or t.value = 'verylow')))
    ->collect(f | if f.tags->exists(t | t.key = 'detection_quality' and
    ↪  t.value = 'low') or f.tags->exists(t | t.key = 'detection_quality'
    ↪  and t.value = 'verylow') then 0.702 else 0 endif)
    ->sum() + 0.00787
in
'the latency of the system is ' + totalEnhancement.toString() +  's'
```

Figure 7: OCL code of the Latency query in the Smart Parking use case (real return value printed in plain readable text).

language, and 2) the Object Constraint Language (OCL)[5] to show the compatibility of our language with a standard navigation model and query language. The final reporting on the query execution results is currently implemented as strings to be displayed onto the Eclipse workbench's console, or to be printed into textual files to be then shared with the Fog System Architects.

All the resources described in the paper, including the complete source code of the proposed Fog Modeling Language, the models of the different use cases and the implementation of the related queries, are available in an open repository[6].

7 DISCUSSION

7.1 Current scope and limitation

The presented work capitalize on existing work in the context of Cloud systems. However, as seen in the literature, there are significant differences between Cloud and Fog systems. In fact, the Edge and IoT layers of Fog systems have particular types of elements and properties which are not present in traditional Cloud systems (e.g., IoT devices or material at the edge with their hardware and software properties, heterogeneous communication means). Notably, the concept of Fog Area is key to model several subsystems supporting the decentralized characteristic of Fog systems. Moreover, characterizing network types and links is fundamental in Fog systems to accurately model the communications between the different Fog areas and their elements. Finally, the modeling of applications and offered services is also important in the context of highly distributed Fog systems. Up to our current knowledge, existing approaches and languages do not allow to properly model the large variety of Fog systems, specify their characteristics, and express related non-functional properties to be verified. Our generic and customizable approach is a direct answer to this challenge.

VeriFog and FML are generic because we can model different, and potentially any, kinds of heterogeneous Fog systems within different applications domains while considering different types of non-functional properties to be verified. This notably includes properties that are more critical in Fog systems than in Cloud systems (e.g., security or performance/latency issues). It is important to note that our intent is not to support by default in our approach all the possible Fog-specific issues. For example, the resilience to faults is deliberately not a 'first-class' concept in FML. This is why we designed our approach as customizable so that 1) FML can be refined to add the needed attributes/metadata thus enriching the Fog system models as required and 2) additional queries can be defined accordingly to verify any issues (e.g., fault tolerance) thanks to these complementary attributes/metadata. Finally, even though the usage of our approach in this paper focuses on the design time verification of non-functional properties over Fog systems, we also plan to work on its usage in other steps of the Fog system's life cycle (cf. Section 9). This way, we will further explore the support for

[5]https://projects.eclipse.org/projects/modeling.mdt.ocl
[6]https://zenodo.org/record/8104709

other different Fog-specific issues that cannot be easily addressed at design time (e.g., more runtime related issues such as reliability).

7.2 Lessons learned

We had very frequent interactions with our partner company Smile regarding the industrial applicability of our work.

From the positive side, we demonstrated in practice to them that we can model different kinds of Fog systems in different application domains while considering different types of non-functional properties to be verified (e.g., security). This was perceived as very promising by Smile, especially since it opens the door for more interesting usages of our approach and corresponding FML models in the context of the other steps of the Fog system's life cycle (cf. Section 9). Indeed, Smile is interested in providing in the future more complete DevOps pipelines supporting the design, development, and execution of such large-scale distributed systems. The present work is a relevant starting point for such future experiments and related evaluation at a larger scale (e.g., with more engineers).

From the less positive side, it is currently quite challenging in industry to find such a FSA having both a solid system architectural/modeling expertise (design part) and sufficient technical/programming skills (verification part) for Fog systems. Notably, Smile stated that the typical DevOps engineer is usually not very familiar with modeling languages in general. To reduce this gap, we prototyped FML with a YAML concrete syntax that is quite close to the kind of configuration files the DevOps engineer usually handles. Nevertheless, a dedicated modeling training would still be required, and the learning curve should not be underestimated. To possibly overcome this, additional means of (re)using FML models could be envisioned. This could be realized via dedicated graphical interfaces, possibly integrated into existing DevOps solutions, for example to replace OCL (as mostly known in the MDE community) and Java (sometimes too general-purpose and verbose) in VeriFog.

Ultimately, Smile believe that the FSA profile will emerge when Fog Computing technologies will become mainstream and widely used in different application domains. In the meantime, in practice, the role of the FSA can also be realized by a team of engineers: a design specialist can take care of modeling the system while a security expert (for instance) can deal with corresponding verification aspects. In this case, both will have to frequently interact together and FML could be a good facilitator when used as a commonly shared language for modeling Fog systems.

8 RELATED WORK

8.1 Simulation and emulation

Designing, deploying, and evaluating Fog-based applications is a complex and costly endeavor. To this end, several works proposed to rely on simulation and emulation tools associated to dedicated languages. For example, Fogify [26] provides a modeling language for defining Fog topologies (encapsulating QoS constraints) which extends the Docker-compose specification. However, contrary to our approach, it does not directly cover the modeling of configurations and mostly focuses on Docker-based infrastructures. iFogSim [15], based on CloudSim [11], provides a modeling language for IoT devices and associated Fog resources, in order to enable the simulation of scheduling policies based on multiple QoS criteria. In this case, contrary to us, they have very specific runtime objectives. Another example is EmuFog [22] that enables the from-scratch specification of Fog Computing infrastructures and the emulation of real applications and workloads. Once again, they are mostly operating at runtime and with a specific target in mind (i.e., workload management). Even if all these work intend to somehow verify Fog systems before their deployments, they are closely related to particular industrial tools (e.g., Docker) or simulators (e.g., CloudSim). Moreover, they are not promoting a more generic modeling language to be exploited at each phase of the system's life cycle (cf. Section 9).

8.2 Quality of Service (QoS)

There are also a few model-based solutions targeting the QoS of Fog systems. For example, FogTorch [8] proposes a semi-formal language that considers various relevant Fog aspects in order to determine QoS-aware deployments of IoT application. Its successor FogTorchΠ [9] exploits Monte Carlo simulation models to take into account possible variations of the QoS and eligible deployments of Fog applications. SMADA-Fog [24] provides a semantic model-driven approach to support the deployment and adaptation of container-based applications in Fog Computing. The used language relies on two metamodels implemented within the Node-RED[7] deployment tool. However, in all cases, these solutions are strongly deployment phase-oriented, and do not target an end-to-end holistic approach as we do (cf. Section 9).

8.3 Life cycle management and orchestration

As introduced before, the design and implementation of a complex Fog system can quickly become really challenging. Some works partially address the life cycle of Fog systems in order to overcome this complexity [14, 29]. However, they mainly deal with the execution phase and are not necessarily meant to be generalized to the support for the whole life cycle, notably since they do not come with reusable modeling languages. Indeed, a recent survey [12] found out that orchestration is an over-used word that sometimes refers to life cycle management. This semantic shortcut unfortunately leads the community to a too strong focus on runtime concerns. In fact, this survey shows that Fog orchestrated entities, when properly modeled as services, tasks, pipe-lines, workflows, etc. could be used more intensively in the context of Fog life cycle management.

From an industrial perspective, there are also relatively recent initiatives. For example, Fogernetes [28], based on Kubernetes[8], compares and maps the requirements of application components to available Fog nodes in order to ensure an optimal deployment according to some non-functional properties. However, this solution specifically relies on the Kubernetes, and focuses on non-functional properties while we could also possibly support functional properties as well (even if this is not the focus of the work presented in this paper). Going further, GitOps [7] is one of the main trends in the DevOps [17] ecosystem for Continuous Deployment that promotes infrastructure automation in highly distributed applications. A recent work [19] describes an initial implementation of GitOps at the Edge/IoT-level based on KubeEdge[9]. Such an approach could

[7] https://nodered.org/
[8] https://kubernetes.io
[9] https://kubeedge.io

be interesting to generalize in the context of other similar technical frameworks, by relying on FML as a basis for instance.

9 CONCLUSION AND FUTURE WORK

The work described in this paper belongs to a more global research effort intending to support the complete life cycle of Fog systems. In fact, the presented design-time modeling support for the verification of Fog systems is only a first step towards a wider usage of the proposed FML. The longer term objective is to generalize the (re)use of FML models in order to improve the support for other major activities within the system's life cycle. For example, at development time, we would like to consider the FML models as inputs for semi-automatically generating different wrappers or configuration files for various technical platforms (e.g., frameworks, schedulers) and kinds of Fog resources (e.g., Cloud servers, IoT devices). At deployment time, we also plan to rely on FML models in order to partially automate the allocation and/or provisioning of different Fog resources, the chaining of hosted services, the loading/unloading of related tasks, etc. At execution time, we already envision the use of FML models as a way to allow the self-adaptation, in some relevant cases, of the modeled systems (or at least parts of them). For example, one important goal is to make the systems more resilient to various types of faults or errors, or more respectful of high level properties defined in the models.

Another important objective, in direct collaboration with our industrial partner Smile, is to work on the integration of the proposed VeriFog approach, Fog Modeling Language, and other related contributions into real DevOps CI/CD pipelines. Such pipelines can be used in practice to support the production, maintenance and evolution of real Fog systems in industrial contexts, in a managed, offline, way. From an industrial perspective, VeriFog, FML and the corresponding global research effort appear to be a promising way to improve the overall QoS of the target Fog systems while possibly reducing the associated development and management costs.

ACKNOWLEDGMENT

The work presented in this paper was partially funded by the French Agence Nationale de la Recherche (ANR) under grant ANR-20-CE25-0017 (i.e., the SeMaFoR project).

REFERENCES

[1] Mohammad Aazam, Marc St-Hilaire, Chung-Horng Lung, and Ioannis Lambadaris. 2016. MeFoRE: QoE based resource estimation at Fog to enhance QoS in IoT. In *ICT 2016*. IEEE, New York, U.S.A., 1–5.

[2] Zakarea Al-Shara, Frederico Alvares, Hugo Bruneliere, Jonathan Lejeune, Charles Prud'Homme, and Thomas Ledoux. 2018. CoMe4ACloud: An end-to-end framework for autonomic Cloud systems. *Future Generation Computer Systems* 86 (2018), 339–354.

[3] Abdelghani Alidra, Hugo Bruneliere, Hélène Coullon, Thomas Ledoux, Charles Prud'Homme, Jonathan Lejeune, Pierre Sens, Julien Sopena, and Jonathan Rivalan. 2023. SeMaFoR - Self-Management of Fog Resources with Collaborative Decentralized Controllers. In *SEAMS 2023*. IEEE, Melbourne, Australia, 7 pages.

[4] Abdelghani Alidra, Hugo Bruneliere, and Thomas Ledoux. 2023. A Feature-based Survey of Fog Modeling Languages. *Future Generation Computer Systems* 138 (2023), 104–119.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. 2010. A View of Cloud Computing. *Communications of the ACM* 53, 4 (2010), 50–58.

[6] Kamran Sattar Awaisi, Assad Abbas, Mahdi Zareei, Hasan Ali Khattak, Muhammad Usman Shahid Khan, Mazhar Ali, Ikram Ud Din, and Sajid Shah. 2019. Towards a fog enabled efficient car parking architecture. *IEEE Access* 7 (2019), 159100–159111.

[7] Florian Beetz and Simon Harrer. 2021. GitOps: The Evolution of DevOps? *IEEE Software* 39, 4 (2021), 70–75.

[8] Antonio Brogi and Stefano Forti. 2017. QoS-aware deployment of IoT applications through the fog. *IEEE internet of Things Journal* 4, 5 (2017), 1185–1192.

[9] Antonio Brogi, Stefano Forti, and Ahmad Ibrahim. 2017. How to Best Deploy Your Fog Applications, Probably. In *ICFEC 2017*. IEEE, New York, U.S.A., 105–114.

[10] Hugo Bruneliere, Zakarea Al-Shara, Frederico Alvares, Jonathan Lejeune, and Thomas Ledoux. 2018. A Model-based Architecture for Autonomic and Heterogeneous Cloud Systems. In *CLOSER 2018*. SciTePress, Setubal, Portugal, 201–212.

[11] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience* 41, 1 (2011), 23–50.

[12] Breno Costa, Joao Bachiega, Leonardo Rebouças de Carvalho, and Aleteia P. F. Araujo. 2022. Orchestration in Fog Computing: A Comprehensive Survey. *ACM Comput. Surv.* 55, 2, Article 29 (jan 2022), 34 pages.

[13] Resul Das and Muhammad Muhammad Inuwa. 2023. A Review on Fog Computing: Issues, Characteristics, Challenges, and Potential Applications. *Telematics and Informatics Reports* 10 (2023), 100049.

[14] Francesc Guim, Thijs Metsch, Hassnaa Moustafa, Timothy Verrall, David Carrera, Nicola Cadenelli, Jiang Chen, David Doria, Chadie Ghadie, and Raül González Prats. 2022. Autonomous Lifecycle Management for Resource-Efficient Workload Orchestration for Green Edge Computing. *IEEE Transactions on Green Communications and Networking* 6, 1 (2022), 571–582.

[15] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. 2017. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience* 47, 9 (2017), 1275–1296.

[16] Mostafa Haghi Kashani, Amir Masoud Rahmani, and Nima Jafari Navimipour. 2020. Quality of service-aware approaches in fog computing. *International Journal of Communication Systems* 33, 8 (2020), e4340.

[17] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A survey of DevOps concepts and challenges. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–35.

[18] Shancang Li, Li Da Xu, and Shanshan Zhao. 2015. The Internet of Things: a Survey. *Information systems frontiers* 17 (2015), 243–259.

[19] Ramón López-Viana, Jessica Díaz, and Jorge E Pérez. 2022. Continuous Deployment in IoT Edge Computing: A GitOps implementation. In *CISTI 2022*. IEEE, New York, U.S.A., 1–6.

[20] Mukhtar ME Mahmoud, Joel JPC Rodrigues, Kashif Saleem, Jalal Al-Muhtadi, Neeraj Kumar, and Valery Korotaev. 2018. Towards energy-aware fog-enabled cloud of things for healthcare. *Computers & Electrical Engineering* 67 (2018), 58–69.

[21] Keith Massey, Nadia Moazen, and Talal Halabi. 2021. Optimizing the allocation of secure fog resources based on qos requirements. In *CSCloud 2021 / EdgeCom 2021*. IEEE, New York, U.S.A., 143–148.

[22] Ruben Mayer, Leon Graser, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. 2017. EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures. In *FWC 2017*. IEEE, New York, U.S.A., 1–6.

[23] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos. 2018. A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges. *IEEE Communications Surveys & Tutorials* 20, 1 (2018), 416–464.

[24] Nenad Petrovic and Milorad Tosic. 2020. SMADA-Fog: Semantic model driven approach to deployment and adaptivity in fog computing. *Simulation Modelling Practice and Theory* 101 (2020), 102033.

[25] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.

[26] Moysis Symeonides, Zacharias Georgiou, Demetris Trihinas, George Pallis, and Marios D Dikaiakos. 2020. Fogify: A fog computing emulation framework. In *SEC 2020*. IEEE, New York, U.S.A., 42–54.

[27] William Tichaona Vambe, Chii Chang, and Khulumani Sibanda. 2020. A review of quality of service in fog computing for the internet of things. *International Journal of Fog Computing (IJFC)* 3, 1 (2020), 22–40.

[28] Cecil Wöbker, Andreas Seitz, Harald Mueller, and Bernd Bruegge. 2018. Fogernetes: Deployment and management of fog computing applications. In *NOMS 2018*. IEEE, New York, U.S.A., 1–7.

[29] Renchao Xie, Qinqin Tang, Shi Qiao, Han Zhu, F. Richard Yu, and Tao Huang. 2021. When Serverless Computing Meets Edge Computing: Architecture, Challenges, and Open Issues. *IEEE Wireless Communications* 28, 5 (2021), 126–133.

[30] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. 2019. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture* 98 (2019), 289 – 330.