SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 16, No. 1, February 2018, 21-44

UDC: 004.738.5:004.2 **DOI: https://doi.org/10.2298/SJEE1901021P**

Model-based Approach for Semantic-driven Deployment of Containerized Applications to Support Future Internet Services and Architectures

Nenad Petrović[1]

Abstract: Due to increasing number of connected Internet of Things (IoT) devices, enormous amount of data needs to be transmitted to the Cloud for processing, while the network is becoming Cloud computing's bottleneck. On the other side, the privacy and security issues in more sensitive application domains could dramatically restrict the freedom of data movement, so it is not possible to offload all the data to the Cloud for processing. Furthermore, the manual operations related to tuning and deployment of these applications are time-consuming and require additional effort. In this paper, a model-based framework for automated, semantic-driven (re-)deployment of containerized applications is presented, leveraging the synergy of Virtual Network Functions (VNFs) and SDN, tackling the mentioned issues.

Keywords: Container-based virtualization, DevOps, Docker, Edge computing, Future Internet, IoT, Model-driven engineering, NFV, Ontology, SDN, Semantic technology, VNF.

1 Introduction

In recent years, the number of connected devices generating and exchanging data has dramatically increased by introduction of Internet of Things (IoT), pervasive and Edge Computing. The operations related to computing infrastructures and network management have become more challenging, due to large amount of generated data, device mobility and new types of services [1, 2].

Despite the rapid evolution of the data-processing speed, the bandwidth of the network that carries data to and from the Cloud has not increased appreciably [3]. In traditional Cloud computing, the data is offloaded from Edge devices to the Cloud for processing. Thus, with Edge devices generating more data, the network is becoming Cloud computing's bottleneck. In these cases, the processing time of time-critical applications is often limited by the network

[1]Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia;
E-mail: nenad.petrovic@elfak.ni.ac.rs

delay [3, 4]. Also, when we have a large number of devices sensing the environment, uploading the data they generate, it would cause additional network congestion, increasing the network delay, even further. Therefore, there is a need for evolution of network infrastructure management paradigm, which would suit better the novel use cases and services with large number of portable and wearable devices involved, sensing, generating the data about the environment, taking the corresponding action based on decisions brought as a result of data processing and analysis.

The concept of enabling a computer to sense information without any human intervention and act accordingly has been applied to variety of sectors, from home entertainment, transportation, product manufacturing, environmental engineering to healthcare and aerospace engineering. Due to such variety, the second major issue appears. Keeping the privacy and dealing with security of the information in more sensitive domains is crucial. In most cases, it is regulated by law, legal regulations and policies (such as GDPR[2]) that could dramatically restrict the freedom of data movement [3, 4]. Due to laws and regulations, in some cases the data is not allowed to leave the boundaries of the institution storing the data. Thus, in these cases, the data has to be processed within the Edge, instead of being uploaded over vulnerable network to the Cloud.

A possible solution to tackle the mentioned issues is to move the data processing closer to the data generators and consumers [4, 5]. In order to achieve this, it is necessary to enable the application flexibility, so the computation task movement can be performed without affecting the functional aspects of the original application [5].

Therefore, the deployment, monitoring and configuration of such applications is becoming quite challenging. Due to mentioned issues, these operations are more complex and time-consuming, especially if done manually. According to the current trends in software development methodology, such as DevOps, development and operations are tightly integrated in order to provide fast, flexible development and high-quality deliveries in close alignment with business objectives. For this reason, there is a need for a high degree of automation when it comes to the mentioned operations [6, 7].

In this paper, we focus on issues of management, deployment and configuration of container-based applications in order to deal with challenges of new services and use cases. For this purpose, novel concepts and technologies are explored: container-based virtualization, Software-Defined Networking (SDN) and Network Function Virtualization (NFV). As an outcome of the research, we propose a model-driven, semantic-enabled framework for flexible,

[2] https://eugdpr.org/

automated deployment of container-based applications leveraging the synergy of SDN and NFV in order to improve their performance. Principles of model-driven software engineering are used in order to determine the structure and rules for definition of deployment diagrams in design-time, while the semantic reasoning is utilized to draw conclusions that could lead to refined deployment with purpose of performance improvement based on knowledge obtained during both the design- and run-time. In design-time, the topology of deployment diagrams is analyzed, while network monitoring and performance measurement knowledge is utilized in run-time. We decide to use containerized Virtual Network Functions (VNFs) due to fact that they are lighter compared to fully-fledged virtual machines and are more suitable for novel use cases and services where increased mobility and flexibility are required in order to support scenarios such as service hand over or migration.

2 Related Work

Among the published results, several related works utilizing model-driven engineering and automated code generation have been identified. Moreover, the relevance of this approach has been approved by the fact that there have been several recent Horizon 2020 projects (DICE[3] [8] and DITAS[4] [4, 9], for example) based on it.

A significant work in area of Cloud application modeling languages is CloudML, presented in [10]. The deployment model serves as a starting point for the generation of the infrastructure management code, so the application can be deployed using the corresponding deployment engine. Similar approach was used in this paper. However, our goal was to make a modeling language that is even more intuitive than the one presented in [10] by using the familiar UML deployment diagram-alike annotation and introduce the additional modeling concepts to support the scenarios of Edge computing.

The design of execution environment for computation task movement presented in [5] served as a basis for this research. However, it does not support container data persistence and offers quite limited modeling capabilities, as its modeling tool is based on Node-RED data flow editor.

Solutions presented in [11, 12] offer automated, model-driven deployment of container-based applications. The later also includes scenarios of computation task movement between devices of different computing architectures in IoT systems (x86 and ARM), supporting the scenarios of Edge computing. While both of them utilize automated code generation, they only offer static deployment of a given topology and do not involve the re-

[3] http://diceh2020.eu/

[4] https://www.ditas-project.eu/#

deployment mechanisms for performance improvement based on run-time information, neither support SDN or VNF. Furthermore, they do not offer automatic check whether the provided deployment model complies with data movement regulations or no. The framework presented in [12] does not support container data movement.

In [8], a framework for automated, model-driven deployment of data-intensive Cloud applications with iterative enhancements based on run-time metrics using traditional hypervisor-based virtual machines has been presented. The modeling language used for creation of deployment diagrams in this case is quite similar to one presented in [10]. However, this solution does not leverage SDN and provides quite limited support when it comes to VNF (only simple firewall rule generation). It mainly targets the traditional x86-based servers and does not support deployment to low-power and IoT smart devices.

On the other side, [4, 9] deals with mechanisms for automated deployment of data-intensive applications that have to comply with given data movement policies and constraints with support for both Cloud and Edge computing. While it supports both the computation task and data movement, this solution does not provide full materialization of deployment diagrams and does not benefit from SDN for performance tuning, but is rather focused on data utility assessment, data movement issues and how data movement affects the QoS and application performance.

In all of the mentioned cases, the combination of model-driven approach and automated code generation has shown significant speed-up of deployment procedure compared to manual operations, for both hypervisor- and container-based applications. However, these solutions do not leverage the synergy of VNF and SDN.

However, the solutions proposed in [13, 14] make use of Virtual Network Functions combined with SDN with objective to provide more flexible live video streaming platform which would make the management operations for Telco operators. The VNFs are deployed according to pre-defined QoS parameters by each client and traffic control rules applied to shape the service chain. Similarly, the framework presented in this paper leverages the synergy of VNF and SDN, making the applications more flexible and adaptable to the changes in execution environment and QoS parameters.

3 Background

In this section, the underlying concepts used for development of our framework are presented. For each of them, an overview and role they have within the context of our research are given.

3.1 SDN and VNF

As results of research efforts in networking last few years, two new concepts have emerged: Software Defined Networking (SDN) and Network Functions Virtualization (NFV).

The main idea of SDN is to simplify network hardware and make the network configuration easier, while improving the network control flexibility by separating the system that decides where traffic is sent (control plane) from the system that pushes data packets to specific destinations (data plane) and provides central programmatic access which enables more efficient resource utilization [2, 15]. By centralizing the network intelligence, decision-making within the network is performed on a global view of the network, opposed to traditional approach, where each node has its own view and is unaware of the overall network state. This kind of programmability enables network configuration to be highly automated. Using the SDN controller APIs, it is possible to implement practically any algorithm which may perform the exact traffic routing that suits the concrete scenario. OpenFlow [15] is a communication protocol which gives access to forwarding plane of a network switch or router. It was the first standard supporting the concepts of SDN architecture. In this paper, a Java-based Open Flow controller Floodlight[5] was used.

Virtualization of computing resources has changed the concept of infrastructure management and has led to many benefits, such as reduction of the operational costs and making the deployment procedures more convenient. In last few years, a similar approach has emerged when it comes to network resources. Network Functions Virtualization (NFV) enables replacing the traditional physical network devices with software running on conventional commodity servers [2]. This software implements the network functions (called “Virtual Network Functions”) that were originally provided by the dedicated hardware. This concept provides a way to reduce cost and accelerate the service deployment procedure by decoupling network functions from the dedicated hardware and moving them to virtual servers. It gives ability to Internet Service Providers for incremental service deployment in order to satisfy customers’ demands in short time. One of the most popular platforms for deployment and orchestration of VNFs running inside cloud virtual machines is Cloudify[6] in combination with OpenStack[7]. However, in this research we target the deployment of container-based VNFs within the Edge of the network, which is a use case not covered by Cloudify out of the box. Currently, the supported VNFs within our framework are: switch, router, load balancer, parental control device and firewall.

[5] http://www.projectfloodlight.org/floodlight
[6] https://cloudify.co/
[7] https://www.openstack.org/

NFV is not SDN-dependent. It is possible to implement a virtual network function without any usage of SDN-related concepts. However, leveraging the SDN concepts to implement and manage infrastructure based on virtual network functions can be highly beneficial, as it provides fine-grained traffic control and increases the overall flexibility. Therefore, we can say that these two concepts (SDN and NFV) are complementary [1, 2]. In context of this research, containerized virtual network functions are used as a specific type of deployable tasks, while SDN is used to shape the traffic flow between the devices and the deployed virtual network functions in order to support specific scenarios recognized using semantic analysis within the deployment diagram.

3.2 Container-based virtualization

In recent years, in domain of computing resource virtualization another approach appears – container-based virtualization. Containerization is a lightweight mechanism for isolation of running processes, so their interaction is limited only to their designated resources. This way, a virtual execution environment is created at a software level inside the host machine. Containers have been around in Linux for some period already, but they have recently become popular due to open-source technology named Docker[8].

The whole system becomes more resource-efficient as there is no additional layer of hypervisor, and thus no full operating system which can occupy a lot of storage space and memory for each virtual machine [16]. Therefore, container-based virtualization is much more IoT device-friendly. There is a complete Docker ARM port compatible with Raspberry Pi (that was used as a representative example of ARM IoT devices). Thus, we decided to use Docker containers as computation task abstraction [5], which enables to easily move tasks between Cloud and Edge and between devices with different computing architectures by using computation task Docker image counterparts for each of the considered architectures.

When it comes to data management, Docker containers were originally designed to be stateless. It means that changes made during the container execution are not persistent as in case of traditional virtual machines. For this purpose, the concept of volumes was introduced. Volumes[9] are preferred mechanism for persisting data by Docker containers. They represent host directories mounted to containers. Therefore, inter-host volume management requires additional mechanisms to be implemented and integrated in order enable the persistence in container-based applications.

[8] https://www.docker.com/

[9] https://docs.docker.com/storage/volumes/

Container management, scheduling and orchestration technologies (such as Docker Swarm[10] and Kubernetes[11]) are usually used for deployment and management of large-scale container-based applications and services, as they provide container clustering and replication, which in combination with load balancing mechanisms enables the scalability and fault tolerance features. The underlying container management system used for the implementation in this research is Docker Swarm. There are two types of devices within the container cluster: master node and worker nodes. The Swarm is created at master node which is responsible for container management and service creation. The service creation consists of allocation of the desired container to the right device which had joined the Swarm previously and exposing the selected ports. As a container repository, we use public Docker Hub[12] repositories which should contain two versions of each task – ARM and x86, with same name but different suffix. This way, we ensure that, in case of computation task movement, each device can find the right container (ARM or x86). When it comes to data persistence, a separate private SVN repository was used to keep the volume data for each of the containers. Before the service creation, the corresponding data volume is downloaded to the target device and then mounted to the task container. Each time before the task migration is performed, the container data volume directory is committed to SVN, so it can be later mounted by the other host device that will execute the task. During the execution, all the HTTP requests directed to the considered container-based application are first sent to the Docker Swarm master. After that, the master resolves the Swarm worker device where the service is actually deployed, according to the port used to expose the service.

3.3 Metamodeling

Metamodeling is analysis, construction and development of rules, constraints, models and theories applicable and useful for modeling a predefined class of problems [17]. A metamodel is a model of a modeling language which defines the structure and constraints for a family of models. In context of our framework, a metamodel is used to define the structure of the modeling language that is used for creation of deployment diagrams. The user is able to draw a deployment diagram using the modeling tool, which is a part of the proposed framework. The deployment diagram is materialized once the deployment code is generated and submitted to the deployment platform. The rules and constraints related to modeling in our framework are defined by the metamodel given in Fig. 1.

[10] https://docs.docker.com/engine/swarm/
[11] https://kubernetes.io/
[12] https://hub.docker.com/

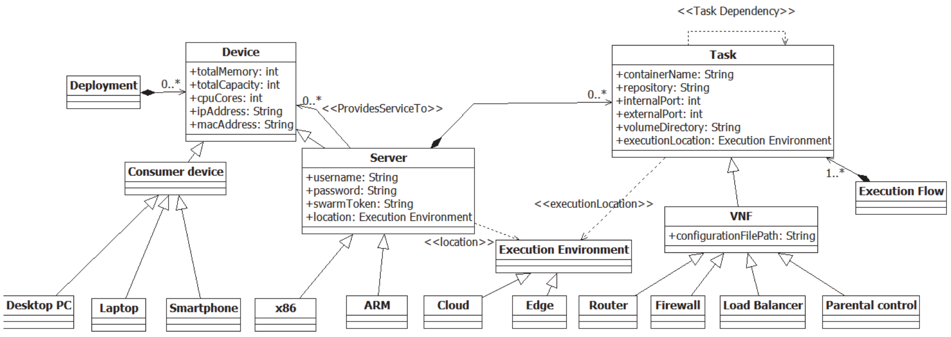

Fig. 1 – *UML diagram of the container-based application deployment metamodel.*

There are two main concepts in our deployment model: device and task. Each deployment diagram created by our modelling tool consists of devices, while some of the devices also contain the allocated tasks.

Devices are split into two categories: consumer devices and servers. Consumer devices are different types of devices (such as traditional desktop PCs, laptops, smartphones, IoT devices) that belong to customers, used for various purposes (from social networking and home entertainment to healthcare applications and enterprise use cases). Servers are devices that belong to the provider that are able to host various applications and services. They can be either traditional x86 or ARM devices.

For each of the consumer devices, we can optionally insert information about IP address, MAC address, number of cores, memory and storage capacity, processor architecture which can be utilized in order to optimize the network deployment, generate additional networking rules or make use of SDN by generating the traffic control rules based on the topology. This kind of information can be especially useful in cases where it is necessary to support Edge computing scenarios. On the other side, the previously mentioned parameters are obligatory for server devices, together with some additional server-specific information, such as username and password (used for SSH), Docker Swarm token (that is used to join the container cluster belonging to a particular Swarm master) and location (Cloud or Edge). Location is of utmost importance as it is used by the mechanisms that provide compliance with given privacy and security policies.

Tasks are abstractions of applications and services offered to customers by service providers. Within the scope of our framework, each of them represents a Docker container which is deployable only to service provider machines. A Docker container is identified by its repository and name within the Docker Hub, which is used as a repository and service discovery. Virtual Network

Functions (VNFs) are specific task type that can be hosted by provider's server device. However, the task can also be any other service (such as video streaming, augmented reality or speech recognition application); it is not necessarily a VNF. As each task corresponds to a Docker container, there are two more parameters: the external port for the service access and internal port used within the Docker host machine. Volume directory is a parameter that specifies the directory within the server that is mounted to the container in order to provide the data persistence utilizing the working principle of Docker volumes. As VNFs are run within the Docker containers, the configuration file path inside the mounted volume can be set for cases when some additional networking rules can be automatically generated which would require the configuration file modification, such as routing table, firewall rules etc. Knowing the right configuration file path gives us also possibility to automatically generate even these rules, store them into a file which will be mounted to the container. Among the VNFs, we have many typical well-known network functions, such as router, load balancer, intrusion detection, parental control, firewall and many others.

Relationship between server and other devices is annotated as <<ProvidesServiceTo>>. This way, we define which consumer devices are under the influence of the deployed VNFs. The devices affected by the virtual network functions could be both consumer devices and other service provider servers. <<TaskDependency>> is a directed association, describing the fact that tasks can be dependent on each other, so the output of one task can be used as the input of another one, which makes an execution flow.

Execution location represents the environment where the considered task could be executed due to data movement constraints, legal regulations, security and privacy policies. According to this, some of the tasks are allowed to be executed only within the Edge of the network, without leaving the physical boundaries of the organization. On the other side, there are tasks whose execution location is not constrained, so they can be executed also in Cloud. Each device can physically reside in Cloud or Edge. Therefore, the metamodel constraint rules could take care of matching execution environment for each of the tasks that have to be deployed. Alternatively, if execution location and server IP address for some task are not specified in the deployment diagram, they will be determined automatically according to the pre-defined policy or a set of constraints.

For the development of our modelling tool, ADOxx metamodeling platform[13] was used. It provides ability to create a full modeling software with graphical user interface based on UML-alike metamodel definition written in ADOxx Definition Language. Additionally, it is also possible to define

[13] https://www.adoxx.org/live/home

modeling rules and constraints. We decided to use ADOxx platform, as it provides an easy and convenient way to automatically construct a complete standalone model editor with visual elements, just by definition of the metamodel which represents the domain of interest.

3.4 Semantic technology

Semantic analysis of a program code is a process of understanding the meaning of the program code based on its context, in a similar way as humans do. However, in order to perform the semantic analysis, it is required both to parse the code and store the knowledge extracted from code in a way that is suitable for reasoning mechanisms, as the knowledge representation formalism used in semantic Web technologies allows logical reasoning that gives ability to infer new information or knowledge from the existing facts.

For this purpose, we use ontologies. Ontology is a formal representation and definition of categories, their properties and relations between concepts, data and entities that substantiate one, many or all domains. RDF[14] is used to formally describe ontologies. It is a data model that provides a way to express simple statements about resources, using named properties and values [18, 19]. In context of RDF, classes are used to define types of things and categories. In addition to defining the classes of things within the ontology, it is also possible to define specific properties that characterize those classes of things. Relations and facts about the classes, their properties and relations are stored as triplets within the RDF triple store for both the ontology definition and its instances. SPARQL[15] is a semantic query language, which is able to retrieve and manipulate the data stored within the RDF triple store [18, 19]. We use the results of SPARQL queries for semantic reasoning, according to the given domain-specific rules, in order to identify specific deployment cases, network states, system conditions and scenarios.

In context of this paper, ontologies are used to represent the knowledge about various design- and run- time aspects. Deployment Model Ontology consists of semantic annotations of the user-created deployment diagrams designed using the modeling tool. While the previously presented metamodel defines the modeling language and static structure of deployment diagram, completely defined at design-time, the Monitoring Data Ontology is used for semantic analysis which includes reasoning mechanisms involving also the knowledge obtained during the run-time beside the design-time aspects. Within our framework, we embed the mechanisms of automatic re-deployment at run-time, according to the system state and changes of condition in execution environment. For this purpose, various aspects are considered: number of

[14] https://www.w3.org/RDF/
[15] https://www.w3.org/TR/rdf-sparql-query/

devices using the service, connection speed, signal strength, various QoS metrics, such as latency and jitter. They are used to express various aspects and consequences related to device mobility and varying network conditions. In Fig. 2, an excerpt from the ontology that captures the run-time knowledge is given.

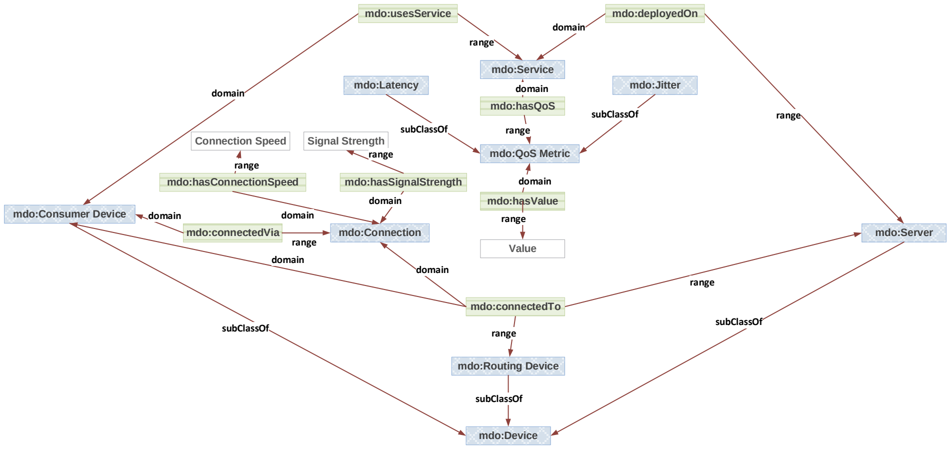

Fig. 2 – *An excerpt from the ontology capturing the run-time knowledge about the deployed services – Monitoring Data Ontology.*

Furthermore, the Policy/Constraint Ontology is used to represent the facts about legal regulations that could constraint the data movement and user-defined policies which are used in order to select the appropriate device for the task deployment, as shown in Fig. 3. It is possible to specify whether the task has to be executed in Cloud, Edge or both and the algorithm which is used to select the device where the task is going to be deployed. Several algorithms that perform the server selection according to some criteria related to their features were implemented, such as fastest CPU, largest memory available, minimal network delay and most consumers connected. While the task execution environment is determined by legal regulations which could constraint the freedom of data movement, the selection of the criteria used to allocate a certain task to the specific device is only up to user.

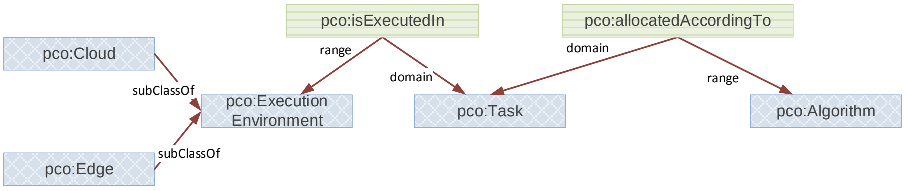

Fig. 3 – *Policy/constraint ontology.*

Device Capabilities Ontology holds the knowledge about the service provider's available devices and their features, important for task allocation mechanisms. An excerpt from this ontology is shown in Fig. 4. For each of the devices, it is necessary to know its computing architecture type, total and remaining memory available, total and remaining storage available, number of CPUs, available network interfaces and location where it resides (Cloud or Edge). Furthermore, the devices can be SDN-enabled, so there is a distinct subclass. Additionally, the devices can contain attached sensors that are taken into account for deployment of specific applications where they are necessary. This knowledge about the available provider's devices is of utmost importance in cases when the exact device that will execute a given task is not known in advance, so the task allocation should be done using automatic mechanisms that also involve device capability matching. Therefore, Policy/Constraints Ontology is used to represent the knowledge that is used to determine the execution location for a task according to the pre-defined constraints and which algorithm should be used for selection policy. On the other side, the Device Capabilities Ontology is responsible for the selection of the exact device that satisfies the given criteria, when it comes to memory available, types of available network interfaces and location where it resides.

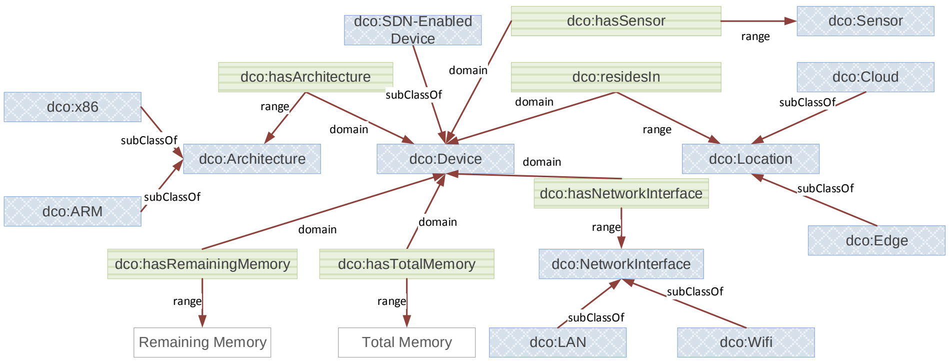

Fig. 4 – *Device Capabilities Ontology.*

In order to perform the semantic analysis, we define domain-specific rules and assertions that are translated to SPARQL queries executed against the RDF triple store which contains the knowledge obtained at both design- and run-time. Based on these results, the reasoning is performed, conclusions are drawn and as an outcome, the code for both the infrastructure and network management is generated according to the actions that have to be taken. The domain-specific rules are defined using the Deployment Rule Ontology, as shown in Fig. 5. Each deployment rule consists of conditions that are going to be tested and actions that have to be taken if the conditions are fulfilled. The conditions can be detected using either design-time (user-drawn deployment diagram) or run-time (monitoring data) knowledge. A SPARQL query is assigned to each of the conditions. According to the given ontology, deployment rules are defined as:

$$rule_i : if\ (d_1 \wedge \ldots d_m) \wedge (r_1 \wedge \ldots r_n)\ then\ action_1 \wedge \ldots action_k$$

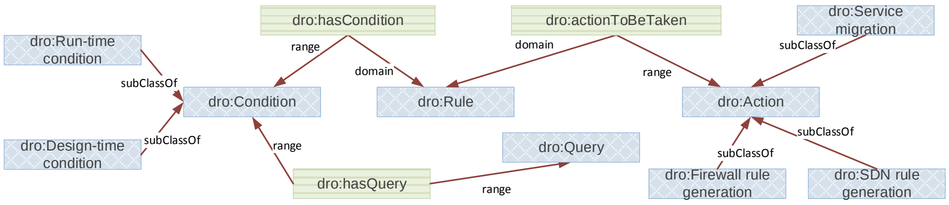

Fig. 5 – *Deployment Rule Ontology.*

4 Implementation overview

In this section, the components of the framework for automated model-based, semantic-driven deployment of containerized applications and the underlying mechanisms are presented (illustrated in Fig 6.).

First, the user draws a deployment diagram using the drag-and-drop graphical interface of the modeling tool environment using the available elements and connecting them. Additionally, the user needs to configure the necessary parameters specific to the concrete type of elements (such as IP or MAC address etc.). Once the modeling is done, the deployment model is exported as XML file written in a domain-specific language.

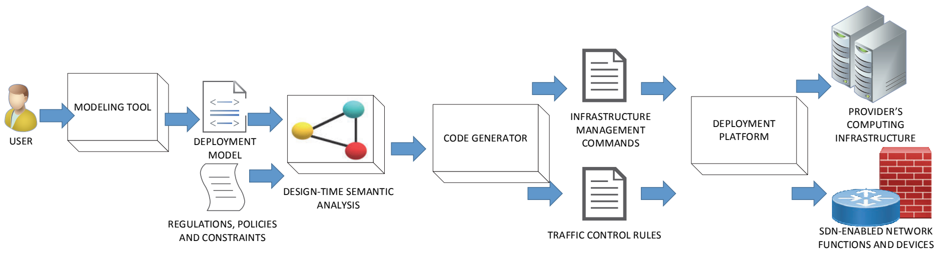
USER
MODELING TOOL
DEPLOYMENT MODEL
REGULATIONS, POLICIES AND CONSTRAINTS
DESIGN-TIME SEMANTIC ANALYSIS
CODE GENERATOR
INFRASTRUCTURE MANAGEMENT COMMANDS
TRAFFIC CONTROL RULES
DEPLOYMENT PLATFORM
PROVIDER'S COMPUTING INFRASTRUCTURE
SDN-ENABLED NETWORK FUNCTIONS AND DEVICES

Fig. 6 – *Overview of the framework for automated model-driven deployment of container-based applications.*

After that, the semantic analysis of the code given in model's domain-specific language is performed. The code written in domain-specific language is parsed and necessary information from the code is inserted as a set of triples into the RDF triple store. The model of the triples that are inserted is defined by the corresponding ontologies. Then, according to domain-specific rules, the reasoning is performed, by executing SPARQL queries against the triple store. These domain-specific rules embed the expert knowledge to treat different use cases where the information from the user-defined deployment diagram could be leveraged in order to optimize the network topology relying on flexibility of SDN. On the other side, the deployment diagram is also checked whether it complies with the pre-defined constraints imposed by legal regulations and policies related to data movement. The reasoning results are consolidated and further processed, so the conclusions can be used by the code generator.

Furthermore, the code generation is performed taking into account both the original deployment model definition and results obtained within the step of semantic analysis. Two types of codes are generated: infrastructure management code (Docker/Docker Swarm or Kubernetes CLI commands) and SDN traffic control OpenFlow rules. Infrastructure management code is responsible for deployment of the desired applications and services in form of containers (both the virtual network functions and others) to the server machines that belong to the service provider. On the other side, SDN rules are used to shape the traffic forwarding within the service provider network. These rules are utilized by SDN-enabled switches and routers, which exist either as virtual instances or real networking hardware. The code generation mechanism is based on the algorithm presented in [12] with extensions related to semantic analysis, SDN rule generation and persistent data management. The results of semantic analysis are used for decision whether it is necessary to generate some specific part of code or no. As one of results, SDN rules are generated in particular scenarios. Moreover, in order to provide container data persistence, the additional commands related to downloading the corresponding image from SVN and volume mounting are also generated.

Finally, both the infrastructure management commands and traffic control rules are submitted to the deployment platform. The deployment platform reads the generated script and executes the corresponding commands in order to allocate the desired containers to provider’s computing infrastructure using the targeted underlying container orchestration and management system, while the SDN controller shapes the network traffic through the provider’s SDN-enabled network functions and devices, according to the generated traffic control SDN rules.

Furthermore, the deployment platform also includes the network status and performance monitoring component, which records the information of the current network state and quality of service (such as number of users connected, utilization of resources, connection/signal strength, speed, delay, number of connected users). This data is filtered, processed and analyzed in order to extract the run-time knowledge. The knowledge obtained as result of data analysis is stored within the RDF triple store. Therefore, it is possible to execute

SPARQL queries and perform reasoning about the run-time conditions. This way, the run-time knowledge is used to perform fine-tuning and re-deployment of the deployed services according to the changes of network state and run-time execution environment. The run-time semantic analysis by the deployment and monitoring platform can be performed periodically or activated by triggers, in order to detect network condition or topology changes which can lead to new code generation, re-deployment or configuration modifications with goal to increase the performance and quality of service of the running applications. Fig. 7 shows the detailed structure of the deployment platform with embedded monitoring, data mining and run-time semantic analysis components.

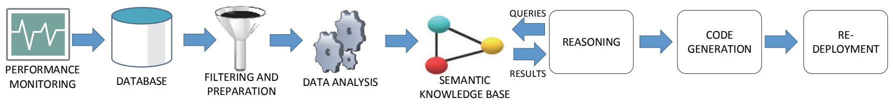

Fig. 7 – *Working principle overview of the deployment platform with embedded mechanisms for run-time semantic analysis.*

4 Scenarios

In this section, three usage scenarios are presented illustrating how both service consumers and providers can benefit from the proposed approach. The first scenario presents an example of design-time semantic analysis. The second scenario is about the deployment under the execution environment constraints due to legal regulations and policies. Finally, the third scenario illustrates the possible benefits of re-deployment based on run-time semantic analysis.

5.1 Design-time semantic analysis: Parental control service

Let us assume that a family that consists of parents and small children uses internet connection provided by ISP at home. However, the parents want to restrict the access, so that the children would not be able to watch the content that is not suitable for their age (such as movies and video games which contain violence). On the other side, the access to the content from other devices should not be affected. Moreover, the assumption is that the internet is accessed using the virtual router hosted on provider's server.

In this case, it would be convenient for the service provider to offer a service of parental control for additional monthly fee and deploy it just by running the parental control application container on their server. This way, the installation, setup and networking hardware cost would be significantly reduced.

First, the deployment diagram is created by our modeling tool. The parental control element is added to the server and all the devices that belong to the children are connected to the server via <<ProvidesServiceTo>> relationship.

For the provider’s server, it is necessary to provide IP address, while for children’s devices (phone and tablet) MAC address should be provided.

Once the deployment model is completed, the automatic code generation process can be launched. The infrastructure management code for the deployment of the parental control service is generated and the corresponding container should be deployed on the target provider’s server. Furthermore, the content filtering rules are generated only for consumer devices whose MAC addresses belong to the given set, submitted to the SDN controller and applied, while all other devices remain unaffected. Furthermore, the semantic analysis of the deployment model code is performed to retrieve the devices which should be under the influence of parental control by executing a SPARQL query.

Finally, for each of the retrieved devices connected to the server where parental control service is deployed, a content filtering rule is generated and applied to the SDN controller in order to take an effect.

5.2 Deployment under execution environment constraints: Infraction test

In this scenario, we are considering the deployment of infraction test application that is used within the medical information system. The application consists of several tasks, packed as Docker containers, each of them running a PHP or database server (MySQL or MongoDB) and performing some data-related operations: data storage/retrieval or processing. There are three types of data processing tasks involved: ECG image analysis, troponin result analysis and infraction test. ECG image analysis takes an ECG image as input in order to determine whether it contains anomalies or no. Troponin result analysis determines whether the patient’s troponin presence in blood is above the given threshold. Finally, the infraction test gives the answer whether the patient had infraction or no. If both the ECG image contains anomalies and troponin is above the given threshold, then the answer is “yes”. Otherwise, the answer is “no”. Fig. 8 shows the structure of the infraction test application.

Table 1 shows an overview and description of tasks that were used to construct the infraction test application. Each of the tasks actually corresponds to a separate Docker image. The first column presents a name of the task. The second column contains the name of the technology used by the task. The third column (environment) tells if the task has to be executed in Cloud or within the Edge of our network during the experiment. The last column indicates the computing architecture of the device executing the considered task.

The deployment model was created using our modeling tool, while the execution environment constraints were inserted into knowledge base, according to the Policy/constraint ontology. For each of the tasks present in the deployment model, the execution location was determined using the design-time semantic analysis.

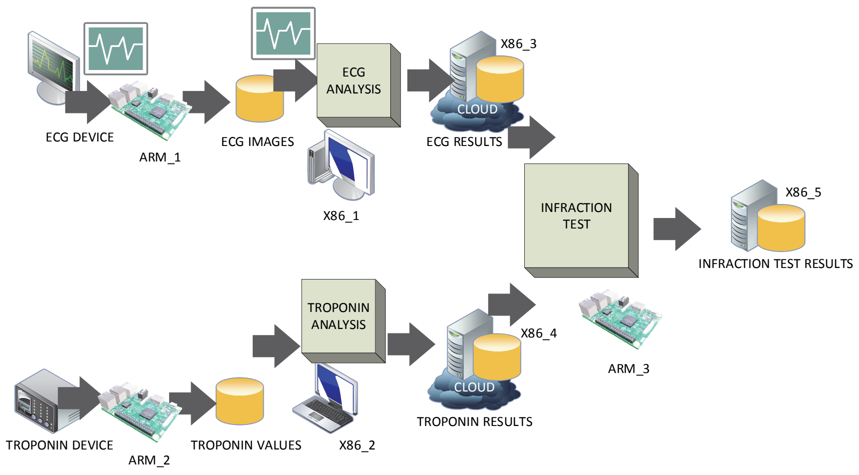

Fig. 8 – *Infraction test medical application overview.*

Furthermore, it is needed to perform the device capability matching according to the location where they reside. The SPARQL query executed in this case returns the possible task-device mapping by matching the task execution environment with device location. After that, the Docker infrastructure code is generated and submitted to the master for service creation.

Table 1
Infraction test application task overview.

Task Name	Tech.	Environment	Architecture
ECG read data	MongoDB	Edge	ARM
ECG processing	PHP	Edge	ARM
ECG write data	MongoDB	Cloud	X86
Troponin read data	MongoDB	Edge	ARM
Troponin processing	PHP	Edge	X86
Troponin write data	MongoDB	Cloud	X86
ECG read result	MongoDB	Edge	ARM
Troponin read result	MongoDB	Edge	ARM
Infraction check	PHP	Cloud	X86
Result storage	MySQL	Cloud	X86

5.3 Run-time service re-deployment: Augmented reality application

Let us assume that museum offers a mobile application which enhances the user experience during the visit by providing real-time augmented reality elements. For example, the user scans a QR beside the artifact, and the rotation-enabled animated 3D model appears near QR code and the spoken story about the observed museum artifact begins. However, in this case the latency could be critical as mobile device would have to download the 3D models, voice files, display the model in the right position and handle the rotation commands given by user. In this case, the museum offers an Edge server which would provide faster download of the content and processing necessary for the 3D model rendering and animation, while the mobile device itself would just record with the camera and send the stream to the Edge server and display the animation results, eliminating this way the overhead introduced by 3D rendering and image processing. Furthermore, there could be many candidate servers, located in different parts of the museum where the application can be deployed. In our case, there were six users and four candidate servers. This scenario illustrates the flexibility of the applied approach to application deployment and ability to adapt to environment changes in case when the server for deployment is selected according to the "most consumers connected" algorithm.

In the beginning, the augmented reality container is deployed to the server, near the entrance of the museum. However, during the museum tour, the visitors move from one place to another (as illustrated in Fig. 9). At one point the connection to the augmented reality server becomes weak, causing the network drops and reduction of the quality of service. In this case, the results of run-time semantic analysis of the network monitoring data will indicate that the quality of service has dramatically dropped. A SPARQL query that returns the number of connected devices to each server is used in this case. The augmented reality service running on server near entrance will be migrated to server with the largest number of visitors nearby, based on connection signal strength. Also, the SDN rules will be modified in order to support the migration of the service.

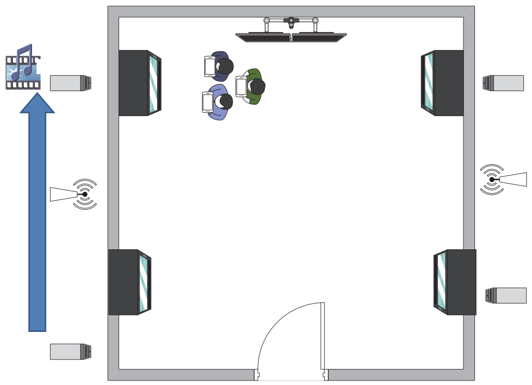

Fig. 9 – *Run-time re-deployment scenario.*

5 Evaluation

In this section, the performance evaluation of proposed framework is presented. The time needed to deploy a given application using our platform and tools for automated deployment was compared against the time needed for the deployment of the same application, if done manually.

The applications from the previously mentioned scenarios where used during the experiments. For evaluation purposes, a laptop equipped with Intel i7 7700HQ CPU and 16GB of DDR4 RAM was used as a Docker Swarm master running the deployment platform and SDN controller in Ubuntu Linux. The same machine was also used for the execution of SPARQL queries, semantic analysis and code generation. The backend of the deployment platform was written in Java.

Table 2
Evaluation results.

Scenario	Design [s]	Semantic annotation [s]	Reason. [s]	Code gen. [s]	Deploy. [s]	Overall auto deploy. [s]	Manual deploy. [s]	Speed-up [times]
Parental control	34	4.57	3.12	7.51	19.44	68.64	220	3.21
Infraction test	58	5.21	4.23	8.59	14.87	90.9	560	6.16
AR app.	21	2.17	2.52	2.76	18.13	46.58	170	3.64

In **Table 2**, the evaluation results are presented. The first column presents the scenario name. The second column shows the time needed for user to draw a deployment diagram of the corresponding application using our modeling tool. The third column is the time needed to parse and semantically annotate the deployment diagram or other data (such as network/performance monitoring stats). The fourth column is the time that is spent for the execution of the corresponding SPARQL queries necessary for the semantic reasoning mechanisms. The fifth column holds the values of time needed for the code generation. The sixth column represents the time that needed for the deployment of the generated infrastructure code and SDN rules. The seventh column shows the sum of previous column values (from second to sixth) and represents the overall time that needed for the automatic deployment procedure – from user-drawn model to fully functional application. The eighth column shows the average time needed for the manual procedure of application deployment without using our platform. The last column shows the speed-up of deployment time as a quotient of manual deployment time (seventh column) and the overall automatic deployment time (sixth column). In all cases, we were considering the "warm deployment time", which means that all the necessary Docker images had already been downloaded. Each value in table is an average of 10 measurements.

6 Discussion

Considering the evaluation results, it is noticeable that the presented approach definitely speeds-up the deployment procedure in considered scenarios using the automatic mechanisms, despite the overhead introduced by semantic code annotation, semantic reasoning and code generation. However, it varies from scenario to scenario. The speed-up is most significant in the second case, as SDN rules were not generated and applied, which reduces the deployment time. Despite the fact that generating SDN rules and applying them to SDN controller slows down the overall deployment procedure, it can lead to performance benefits. It is also observable that the semantic annotation in second scenario lasts more than the others, as the considered deployment diagram was larger than the rest, involving more devices and tasks that need to be traversed during the code generation. Moreover, the time needed for semantic reasoning is also longer, as the deployment

Comparing the code generation and overall deployment time with solutions presented in [11] and [12], it is concluded that this solution is slightly slower, due to fact that previously presented tools only provide static deployment of container-based applications and do not leverage the benefits of Software Defined Networking neither involve any kind of any semantic analysis, which introduces additional overhead. Similar infraction test application case study was presented in [12], where it achieved faster deployment and code generation time. This can be explained by the fact that the solution from [12] does not involve container volume management mechanisms, as less time is needed for deployment without mounting the corresponding volumes. To sum up, the previous solutions do provide greater speed-up than this one, but they lack certain features, such as SDN-enabled networking and container volume management.

On the other side, comparing with another existing solution [8] which is oriented on automated, model-driven deployment of virtual machine-based Cloud applications, we can notice that the relative speed-up obtained in [8] was greater, while the deployment time of our solution is much faster, as expected. This can be explained by the fact that the manual virtual machine setup and configuration generally takes more effort compared to container-based approach. When it comes to absolute deployment duration, it is much quicker in container-based approach, as there is no need to spawn a full operating system (using hypervisor) each time we deploy a task, which is an advantage in favor of container-based systems, especially when we take into account that low-power Raspberry Pi devices are involved. That solution does not provide support for Edge devices with ARM architecture and has quite limited support for virtual network functions and software-defined networking (only the deployment of virtual firewalls and firewall rules).

7 Conclusion and Future Work

The outcome of research presented in this paper is a framework that enables highly automated deployment of VNF/SDN-enabled container-based applications to support future services and architectures that introduces a significant speed-up of the operations, which is of utmost importance in trending DevOps software development methodology. Application deployment time (and so the speed-up) using the proposed framework depends on the concrete scenario - the number of devices involved and services that have to be deployed and also the semantic overhead introduced in specific cases, based on domain-specific rules utilized for semantic analysis. Furthermore, this approach enables the flexibility of applications by providing the capability of computation task movement between the devices with different computing architecture (ARM/x86) and location (Edge/Cloud) in order to comply with pre-defined legal constraints and policies.

The combination of VNFs, SDN and container-based virtualization has great potential for cost reduction as it is not always necessary to rent a full virtual machine for each of the network functions and cuts additional expenses for buying the vendor-specific networking hardware while providing ability of responsive, dynamic quality of service control and fine-tuning at the same time, that could be highly beneficial in Edge Computing use cases.

However, containers are quite new in universe of network function virtualization and there is still a lot of development in progress, as there are many open issues, especially when it comes to security [20]. In container-based virtualization approach, the host’s operating system is exposed to all containers, which introduces potential security issues related to multi-tenancy. Therefore, traditional virtual machines are still ahead from this point of view. Also, the list of available containerized virtual network functions is still quite limited, but the support is about to be extended in future. Despite the existence of security issues, utilizing containers for deployment of virtual network functions seems promising approach, considering the much smaller overhead compared to full-fledged virtual machines which leads to smaller delay, better performance as shown in [21] and also faster deployment time which is crucial in cases of service re-deployment according to the network condition changes with purpose of increasing the quality of service.

For future work, the detailed performance evaluation for the applications using the generated SDN rules is planned (especially when it comes to scenarios that leverage the synergy of VNF and SDN) and extension of the reasoning mechanisms to support novel use cases of future 5G networks, where they are recognized as enabler technologies [22]. Additionally, the application of more sophisticated data analysis techniques to extract the hidden knowledge about the network state and performance from the monitoring data is considered. Furthermore, leveraging the Docker Swarm replication features and implementation of data management mechanisms that will enable persistence of container data for all the running replicas of a task could be highly beneficial, when it comes to scalability and features related to fault tolerance while making the container-based applications more robust.

8 Acknowledgements

This work has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Grant Agreement No 645220, project RAWFIE (Road-, Air- and Water- based Future Internet Experimentation). Moreover, this paper is an extended version of the work published under the name "Model-driven Approach for Deployment of Container-based Applications in Fog Computing" at the 5th international annual conference IcETRAN (Palics, Serbia, June 11–14, 2018), and which was awarded as The Best Young Researcher's Paper at the Computing and Information Engineering section.

9 References

[1] A. C. Baktir, A. Ozgovde, C. Ersoy: How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases, and Future Directions, IEEE Communications Surveys & Tutorials, Vol. 19, No. 4, June 2017, pp. 2359 – 2391.

[2] S. K. N. Rao: SDN and Its Use-Cases - NV and NFV – A State-of-the-Art Survey, NEC Technologies India Limited., White Paper, 2014, pp. 3 – 25.

[3] Z. Mahmood, M. Ramachandran: Fog Computing: Concepts, Principles and Related Paradigms, Fog Computing Concepts, Frameworks and Technologies, Springer, Cham, Switzerland, 2018, pp. 3 – 21.

[4] P. Plebani, D. Garcia-Perez, M. Anderson, D. Bermbach et al.: Information Logistics and Fog Computing: The DITAS* Approach, Proceedings of the CAiSE 2017 Forum Papers, Essen, Germany, 2017, pp. 129 – 136.

[5] N. Petrovic: Enabling Flexibility of Data-Intensive Applications on Container-Based Systems with Node-RED in Fog Environments, Master Thesis, Politecnico di Milano, Milan, Italy, 2017, Ch. 3, pp. 18 – 62.

[6] C. Ebert, G. Gallardo, J. Hernantes, N. Serrano: DevOps, IEEE Software, Vol. 33, No. 3, May-June 2016, pp. 94 – 100.

[7] R. Sturm, C. Pollard, J. Craig: DevOps and Continuous Delivery, Application Performance Management (APM) in the Digital Enterprise: Managing Applications for Cloud, Mobile, IoT and eBusiness, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, USA, 2017, Ch. 10, pp. 121 – 135

[8] Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements, Deployment Abstractions - Final Version, Editors: D. A. Tamburri, E. Di Nitto, 2017, Available at:

http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2017/07/D2.4_Deployment-abstractions-Final-version.pdf

[9] P. Plebani, D. Garcia- Perez, M. Anderson, D. Bermbach et al.: DITAS: Unleashing the Potential of Fog Computing to Improve Data-Intensive Applications, Advances in Service-Oriented and Cloud Computing, Editors: Z. A. Mann, V. Stolz, European Conference on Service-Oriented and Cloud Computing (ESOCC 2017), Springer, Cham, Switzerland, Vol. 824, 2018, pp. 154 – 158.

[10] N. Ferry, H. Song, A. Rossini, F. Chauvel, A. Solberg: CloudMF: Applying MDE to Tame the Complexity of Managing Multi-Cloud Applications, Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing, London, UK, December 2014, pp. 269 – 277.

[11] F. Paraiso, S. Challita, Y. Al-Dhuraibi, P. Merle: Model-Driven Management of Docker Containers, Proceedings of the 9th International Conference on Cloud Computing (CLOUD), San Francisco, USA , June-July 2016, pp. 718 – 725.

[12] N. Petrovic: Model-Driven Approach for Deployment of Container-Based Applications in Fog Computing, Proceedings of the 5th International Conference on Electrical, Electronic and Computing Engineering (IcETRAN 2018), Palics, Serbia, June 2018, pp. 1084 – 1089.

[13] G. Baldoni, M. Melita, S. Micalizzi, C. Rametta, G. Schembra, A. Vassallo: Video Broadcasting Services Over SDN-NFV Enabled Networks: A Prototype, Procedia Computer Science, Vol. 98, 2016, pp. 560 – 565.

[14] A. Gupta, M. F. Habib, U. Mandal, P. Chowdhury, M. Tornatore, B. Mukherjee: On Service-Chaining Strategies Using Virtual Network Functions in Operator Networks, Computer Networks, Vol. 133, March 2018, pp. 1 – 16.

[15] W. Braun, M. Menth: Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices, Future Internet, Vol. 6, No. 2, 2014, pp. 302 – 336.

[16] W. Felter, A. Ferreira, R. Rajamony, J. Rubio: An Updated Performance Comparison of Virtual Machines and Linux Containers, International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, USA, March 2015, pp. 171 – 172.

[17] M. Brambilla, J. Cabot, M. Wimmer: Model-Driven Software Engineering in Practice, 1st Edition, Morgan & Claypool Publishers, San Rafael, USA, 2012 , pp. 13 – 16.

[18] P. Hitzler, M. Krotzsch, S. Rudolph: Foundations of Semantic Web Technologies, 1st Edition, Chapman and Hall/CRC, Boca Raton, London, New York, 2009.

[19] J. Davies, R. Studer, P. Warren: Semantic Web Technologies: Trends and Research in Ontology-Based Systems, 1st Edition, John Wiley & Sons, Ltd., Chichester, England, 2006.

[20] A. Mouat: Docker Security: Using Containers Securely in Production, 2015, Available at:

https://www.oreilly.com/ideas/docker-security

[21] J. Anderson, H. Hu, U. Agarwal, C. Lowery, H. Li, A. Apon: Performance Considerations of Network Functions Virtualization Using Containers, Proceedings of the International Conference on Computing, Networking and Communications (ICNC 2016), Kauai, USA, February 2016, pp. 1 – 7.

[22] M. G. Pérez, G. Martinez Perez, P. G. Giardina et al.: Self-Organizing Capabilities in 5G Networks: NFV & SDN Coordination in a Complex Use Case, Proceedings of the 3rd Network Management Workshop for 5G Networks (EuCNC18), Ljubljana, Slovenia, June 2018, pp. 1 – 5.