From UML Modeling to UPPAAL Model checking of 5G Dynamic Service Orchestration

Ashalatha Kunnappilly
Mälardalen University
Västerås, Sweden
ashalatha.kunnappilly@mdh.se

Peter Backeman
Mälardalen University
Västerås, Sweden
peter.backeman@mdh.se

Cristina Seceleanu
Mälardalen University
Västerås, Sweden
cristina.seceleanu@mdh.se

ABSTRACT

The new 5G technology has the ability to create logical communication networks, called network slices, which are specifically carved to serve particular application domains. Due to the mix of applications criticality, it becomes crucial to verify if the applications' service level agreements are met, especially for the mission-critical scenarios, before the system is up and running. In this paper, we propose a novel framework for modeling and verifying 5G orchestration of dynamic services, which considers simultaneous access of network slices, admission of new requests to slices, virtual network function scheduling, and routing. Due to the dynamic nature of the problem such verification becomes a challenging issue. By combining the benefits of modeling in user-friendly UML, with model checking using UPPAAL, our framework helps to address the issue by enabling both modeling and formal verification at design stage. We demonstrate our approach on a case study that involves: (i) a mission-critical 5G-assisted robot surgery e-health application, accomplished by using a health slice that is simultaneously accessed by various health professionals using a 5G-enabled camera, and (ii) a less critical video streaming application using a video slice, accessed via various 5G-enabled mobile phones, within the same area as the robotic application. By employing our approach, one can verify that the critical health application meets its timeliness requirements, but also that all slices are eventually served in the system.

KEYWORDS

5G, Service Orchestration, Network Slicing, UML, Model Checking

ACM Reference Format:
Ashalatha Kunnappilly, Peter Backeman, and Cristina Seceleanu. 2021. From UML Modeling to UPPAAL Model checking of 5G Dynamic Service Orchestration. In *7th Conference on the Engineering of Computer Based Systems (ECBS 2021), May 26–27, 2021, Novi Sad, Serbia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459960.3459965

1 INTRODUCTION

The fifth generation of wireless technology, 5G, has the potential to support a variety of applications with different requirements, be they low latency, high bandwidth or increased number of connections. This is ensured via its ability to create end-to-end *network slices*, tailored to support respective application requirements [13]. A 5G network slice is composed of several *virtual network functions* (VNFs) that are chained in order to meet the application's functionality. Most often, VNFs have constraints on CPU, RAM, storage, which need to be met by the servers that host them. In addition, servers are connected via links, hence chaining VNFs incurs additional resource overhead in terms of link bandwidth and delay. Adding to the complexity, VNFs can be shared between slices that are requested simultaneously by various 5G user equipment. To analyze if a 5G network slice instance can effectively serve its applications, one needs to ensure that the respective VNFs are allocated, scheduled, and routed according to the current network scenario. This is referred to as **dynamic 5G service orchestration**. For instance, when applications of different criticality share the same network resources one needs to ensure that all slices, especially the mission-critical ones, facilitate meeting application requirements.

Although much research has been devoted to solving the 5G service orchestration problem by providing optimal VNF allocation and routing schemes [12, 17], there is a lack of endeavors that provide modeling and formal verification frameworks that can analyze such schemes early in the design stage, to provide guarantees of the intended system behavior. In this paper, we propose such a modeling and formal analysis frameworkwhich combines user-friendly UML-based modeling [8] with mathematical assurance via exhaustive model checking in UPPAAL [11]. This work builds on our previous results [10], the so-called UML5G-SO framework, that allows one to model and analyze VNF allocation and routing, assuming static worst-case scenarios. In this paper, we augment the UML5G-SO framework to support dynamic system behavior stemming from dynamic slice requests from different user equipment (UE), as well as scheduling, link utilization, etc.Our contribution includes the following: (i) extending the UML5G-SO profile with stereotypes to model UE and the controllers for handling and monitoring the dynamic requests, (ii) defining the behavioral view of a system built based on our profile in terms of restricted UML statechart patterns (see 5.3), (iii) defining pattern-based timed automata semantics for the restricted statechart patterns, to be able to model-check UML5G-SO behaviors with UPPAAL [11], and (iv) implementing tool support for the automatic generation of UPPAAL models from restricted state charts. To demonstrate our approach, we consider a case study of simultaneous access of shared network resources by two applications of different criticality.

The rest of the paper is organized as follows. Sec. 2 details the problem statement and our case study. In Sec. 3, we overview the preliminaries of UML 2.0 modeling, timed automata and UPPAAL

ECBS 2021, May 26–27, 2021, Novi Sad, Serbia

ACM ISBN 978-1-4503-9057-6/21/05…$15.00
https://doi.org/10.1145/3459960.3459965

model checker. Our methodology is explained in Sec. 4. In Sec. 5, we present our extended UML5G-SO profile in UML 2.0 allowing UML modeling of dynamic service orchestration in 5G systems. Sec. 6 presents the formal semantics of the UML model in terms of UPPAAL timed automata. Thereafter, in Sec. 7, we present our G^5tool that allows the automated verification of system requirements, as well as the verification results using the UPPAAL model checker applied on the case study, followed by a brief discussion of the gained insights. We compare our contribution to related work in Sec. 8, before providing the concluding remarks and directions of future work, in Sec. 9.

2 PROBLEM DESCRIPTION

This paper aims at providing a modeling and formal analysis framework for the dynamic 5G service orchestration problem. Concretely, the framework allows a 5G engineer to model an existing service orchestration solution (that is, VNF allocation and routing) of a network slice, and formally verify if the solution meets its application requirements, under various dynamic behaviors assumed as follows:

- *Dynamic network load:* We produce a dynamic network load by using a set of 5G user equipment (UE) that can request a network slice at any point of time.
- *Dynamic VNF scheduling:* We assume that every host executes VNFs (when required) according to a given scheduling policy. The execution time of a VNF is within its best-case and worst-case execution time bounds.
- *Dynamic link utilization:* Instead of assuming that link bandwidth is always reserved for a particular slice, we cater for its dynamic usage, that is, consumption of the respective link bandwidth when used, followed by a subsequent release after its use.

Consider an example of an overlay network consisting of virtual machines (hosts) deployed on edge/cloud servers. We assume that this overlay network is powered by 5G, which supports a variety of applications via end-to-end network slices. A slice consists of a number of virtual network functions (VNFs) interconnected via a VNF Forwarding Graph (VNFFG), assumed a sequence – VNF Forwarding Sequence – in this paper. A virtual network function (VNF) is defined as a software implementation of a network function, which can be easily deployed on virtual resources such as virtual machines [12]. We also assume that the hosts communicate via virtual links, which incurs overheads in terms of bandwidth capacity and latency. In order for the overlay network to serve various applications of different requirements, the network slices' VNFs need to: (i) be allocated on hosts, respecting the latter's processing, memory, and storage capabilities, and (ii) be routed such that the respective VNF chaining is achieved. In our previous work [10], we have already proposed a UML profile called UML5G-SO, and associated static OCL-based analysis of 5G service orchestration solutions, that is, checking whether a given VNF allocation and routing of a slice at a particular point of time meets the application's Quality of Service (QoS) requirements. However, our analysis has considered only static worst-case scenarios, that is, we assume that the system is serving the maximum number of user requests under a maximum load. This is not realistic if one considers the actual varying network load that requires utilizing hosts and links in a dynamic manner.

We assume that each slice has a certain allocation and routing defined (or generated) when the system starts its operation. Our aim is to analyze if the given allocation and routing can meet the application's QoS, considering dynamic behaviors as described above.

To address this, we provide a modeling and formal verification framework that combines UML-based modeling and UPPAAL model checking to analyze 5G-SO systems. To begin with, we present a case study that we model and verify within our framework.

2.1 Case study

As a running example, we consider two applications, namely a robot-assisted surgery application, and a video-streaming application, accessing a health slice and a video slice, respectively, within the same small cell (i.e., bandwidth resources are consumed from the same cell tower). The applications access their respective slices via their 5G user equipment (UE). In our case study, a 5G camera accesses an instance of the *health slice*, and a mobile phone accesses an instance of the *video slice*. We consider that health slices have a higher priority over video slices.

We assume that we have three UE accessing the health slice and two UE accessing the video slice simultaneously, via their slice instances. The case study is depicted in Fig. 1. The slice instances are made of their respective VNF instances. For example, health-slice instances consist of VNF sequence v1-v2, where v1 is an instance of VNF A and v2 is an instance of VNF B. Similarly, a video-slice instance is a VNF sequence v1-v3-v4-v5, where v1, v3, v4, v5 are of category VNF A, VNF B, VNF C, and VNF B, respectively (the slices share some VNFs). The overlay network comprises four virtual hosts onto which the VNFs are allocated. For instance, as shown in Fig. 1, we consider that v1 and v3 are allocated to Host 1. The hosts are connected via *virtual links* as shown in Fig. 1 (e.g., Host 1 and Host 2 are connected via L1). However, not all hosts have a one-to-one link connectivity established, for instance, there is no direct link between Host 1 and Host 3. Hence, any routing scheme that needs to travel from Host1 to Host3 must use the path consisting of links <L1, L2>, or the alternative path <L4, L3>.

For the robotic surgery application established via the health slice, low latency and high bandwidth requirements are of great concern. In comparison, the video streaming application using the video slice has high bandwidth requirements, but not so critical latency constraints. Some of the application requirements that would be interesting to verify are listed below:

Req1: For all possible scenarios, the end-to-end latency requirements of the health/video slices are met, respectively.

Req2: For all possible scenarios, the end-to-end bandwidth requirements of the health/video slices are met, respectively.

Req3: All the instantiated slices in the system are eventually served.

3 PRELIMINARIES

In this section, we introduce the types of *UML diagrams* that we use in this paper, as well as briefly overview *timed automata* and the UPPAAL model checker.

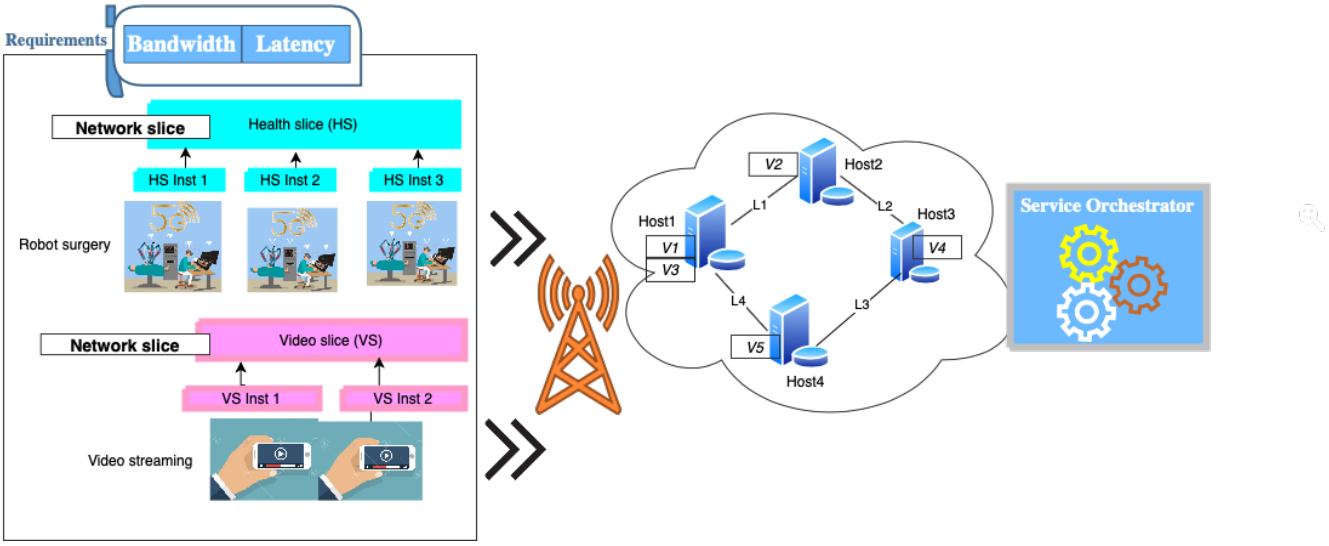

Figure 1: Case study

3.1 UML

The Unified Modeling Language (UML) [8] is a modeling language that helps a designer to express structural and behavioral artifacts of complex systems. In order to capture the essentials of a 5G-SO system, we have to specify its structure and behavior, which we accomplish using our UML5G-SO profile. A *profile diagram* is one of the simplest ways by which one can extend existing UML models, by defining *stereotypes, tagged values* and *constraints* for capturing domain-specific concepts. For a deeper understanding of UML profiles, we refer the reader to relevant literature [8, 9].

In order to model 5G case studies using a UML profile, one should be able to represent both structural and behavioral views of the specific scenarios of the case study, using the profile. In this work, we use UML *class diagrams* and *object diagrams* to represent the system's structure, and UML *statecharts* to represents the system's behavior. While the former are quite straightforward, we will provide a brief explanation of statecharts as they are central to this work.

UML Statecharts. UML statecharts (or UML state-machine diagrams) depict behavior via *states* and *transitions* between states.. While each state simply has a name, a transition includes a *trigger*, a *guard*, and an *action*. The triggers are usually *events*, and the response actions become the effects on the transitions. The *guard* is a Boolean expression that has to evaluate to true in order for the transition to be fired. The UML syntax for a state transition is $Ev(parameters)[G]/A$. In this paper, we chose UML statecharts for their effectiveness of capturing the behavior of individual classes (identified as our system's components) by states, transitions, and transition triggers.

In our work, we consider only two kinds of UML events, *time events* and *call events* [8]. We define a time event (Tm) via the keyword "after", followed by an expression that encodes the time value. We model call events (Cl) as synchronization events that are unicast

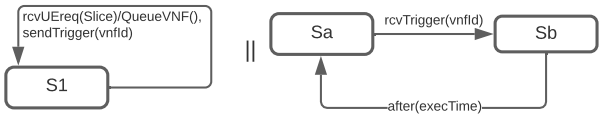

Figure 2: Parallel Composition of UML Statecharts

to other statecharts. The unicast communication offers handshake synchronization between two statecharts, and is blocking, that is, the synchronization takes place only if both sender and receiver are ready to traverse their edges [20]. In addition, we consider that statecharts can be composed in parallel, by defining their interactions via Cl events [20].

Example 3.1. In Figure 2, we show an example of two statecharts that synchronize when executing in parallel. The initial one is the *controller* statechart that has a state S1 waiting for a user equipment (UE) request. On receiving the call event, $rcvUEReq(Slice)$, it queues the respective VNFs to be executed in the host, and generates another call event (action), $sendTrigger(vnfId)$, which is unicast to the *host* statechart. In state *Sa*, the host statechart is ready to synchronize and receives the event $rcvTrigger(vnfId)$, moving to state *Sb*, where it executes the respective VNF and moves back to state *Sa*, when triggered via a time event, modeled by $after(execTime)$.

3.2 Timed Automata and UPPAAL

A Timed Automaton (TA), as used in the model-checker UPPAAL [11], is defined as a tuple, $\langle L, l_0, V, C, A, E, I\rangle$, where: L is the set of finite locations, l_0 is the initial location, V is the set of data variables, C is the set of *clocks*, $A = \Sigma \cup \tau$ is the set of *actions*, where Σ is the finite set of *synchronizing actions*($c!$ denotes the send action, and $c?$ the receiving action) partitioned into inputs and outputs, $\Sigma = \Sigma_i \cup \Sigma_o$, and $\tau \notin \Sigma$ denotes internal or empty

actions without synchronization, $E \subseteq L \times B(C,V) \times A \times 2^C \times L$ is the set of *edges*, where $B(C,V)$ is the set of *guards* over C and V, that is, conjunctive formulas of clock constraints ($B(C)$), of the form $x \bowtie n$ or $x - y \bowtie n$, where $x, y \in C$, $n \in \mathbb{N}$, $\bowtie \in \{<, \leq, =, \geq, >\}$, and non-clock constraints over V ($B(V)$), and $I : L \longrightarrow B_{dc}(C)$ is a function that assigns *invariants* to locations, where $B_{dc}(C) \subseteq B(C)$ is the set of downward-closed clock constraints with $\bowtie \in \{<, \leq, =\}$. Invariants bound the time that can be spent in locations, hence ensuring progress of TA's execution. An edge from location l to location l' is denoted by $l \xrightarrow{g,a,r} l$, where g is the guard of the edge, a is an update action, and r is the clock reset set, that is, the clocks that are set to 0 over the edge. A location can be marked as *urgent* (marked with an U) or *committed* (marked with a C) indicating that the time cannot progress in such locations. The latter is a more restrictive, indicating that the next edge to be traversed needs to start from a *committed* location.

The semantics of TA is a *labeled transition system*. The states of the labeled transition system are pairs (l, u), where $l \in L$ is the current location, and $u \in R^C_{\geq 0}$ is the clock valuation in location l. The initial state is denoted by (l_0, u_0), where $\forall x \in C,\ u_0(x) = 0$. Let $u \vDash g$ denote the clock value u that satisfies guard g. We use $u + d$ to denote the time elapse where all the clock values have increased by d, for $d \in \mathbb{R}_{\geq 0}$. There are two kinds of transitions:

(i) *Delay transitions*: $< l, u > \xrightarrow{d} < l, u + d >$ if $u \vDash I(l)$ and $(u + d') \vDash I(l)$, for $0 \leq d' \leq d$, and

(ii) *Action transitions:* $< l, u > \xrightarrow{a} < l', u' >$ if $l \xrightarrow{g,a,r} l', a \in \Sigma, u \vDash g$, clock valuation u' in the target state (l', u') is derived from u by resetting all clocks in the reset set r of the edge, such that $u' \vDash I(l')$.

UPPAAL. The UPPAAL model checker provides exhaustive model-checking of TA models like the ones overviewed before. A real-time system can be modeled as a network of TA (NTA) composed via the parallel composition operator ("||"), which allows an individual automaton to carry out internal actions, while pairs of automata can perform handshake synchronization. The locations of all automata, together with the clock valuations, define the state of an NTA. The properties to be verified by model checking on the resulting NTA are specified in a decidable subset of (Timed) Computation Tree Logic ((T)CTL) [5], and checked by the UPPAAL model checker. UPPAAL supports verification of liveness and safety properties [11]. The queries that we verify in this paper are of the form: i) **Reachability**: $E\Diamond p$ means that there exists a path where p is satisfied by at least one state of the path, and (ii) **Leads to**: $p \rightsquigarrow q$, which means that whenever p holds, q must hold thereafter, (iii) **Invariance**, $A\Box p$, stating that p should be true in all reachable states for all paths.

4 MODELING AND VERIFICATION FRAMEWORK

In this section, we introduce our proposed framework for modeling and formal analysis of dynamic 5G SO systems. First, we employ our UML5G-SO profile to model the structure and behavior of various 5G-specific case-study scenarios, using UML diagrams, after which we formally verify the latter using the UPPAAL model checker. Our framework has two benefits: (1) a practitioner can use the industrially-accepted UML tool to model the system without any underlying knowledge of the formal TA modeling in UPPAAL, and (2) the verification results obtained by employing UPPAAL provide guarantees to the 5G SO model behavior.

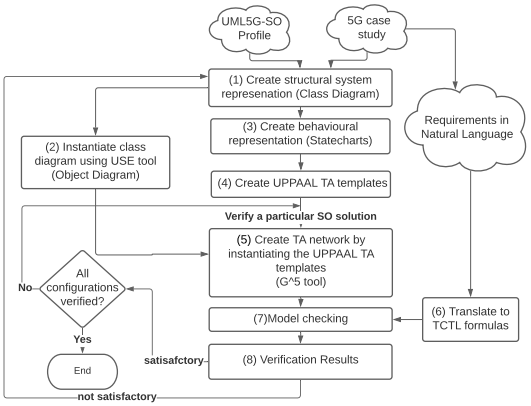

Figure 3: Proposed Methodology

The methodology is illustrated in Fig. 3. Given a case study and its requirements in natural language, we first employ our UML5G-SO profile and create a class-diagram-based structural representation of the case study scenarios. Some of the classes in the class diagram, referred to as active classes, have behavior modeled by UML statecharts. Next, to facilitate modeling, we define statechart patterns and assign semantics to them in terms of UPPAAL TA, hence creating the corresponding UPPAAL TA templates that support creating the formal model counterpart. Last, we formalize the requirements of the case study as TCTL queries (see Sec. **??**, Sec. 6). To verify a particular 5G SO system, we first instantiate the class diagram, yielding an object diagram, to create a run-time representation of the system, for example by using a UML tool such as USE (UML Specification Environment) [2]. Second, we instantiate the TA templates for each active object in the object diagram. Third, we employ model checking, using UPPAAL, against the corresponding TCTL queries. We also provide a tool, G^5, which can execute the second and third step automatically (see Sec. 7.1).

If the requirements are met, we can verify another 5G SO solution, until all the different configurations (VNF allocations and routing) are verified. If some requirement is not met, we need to alter our model and repeat the steps until the former behaves in the intended manner.

5 UML MODELING OF DYNAMIC 5G SERVICE ORCHESTRATION

In order to exemplify how the UML5G-SO profile can be used for modeling 5G-specific scenarios, we detail the modeling of our case study.

5.1 The Enhanced UML5G-SO Profile

The enhanced version of the UML5G-SO profile is shown in Fig. 4 (changes highlighted with colors and in bold). We define three

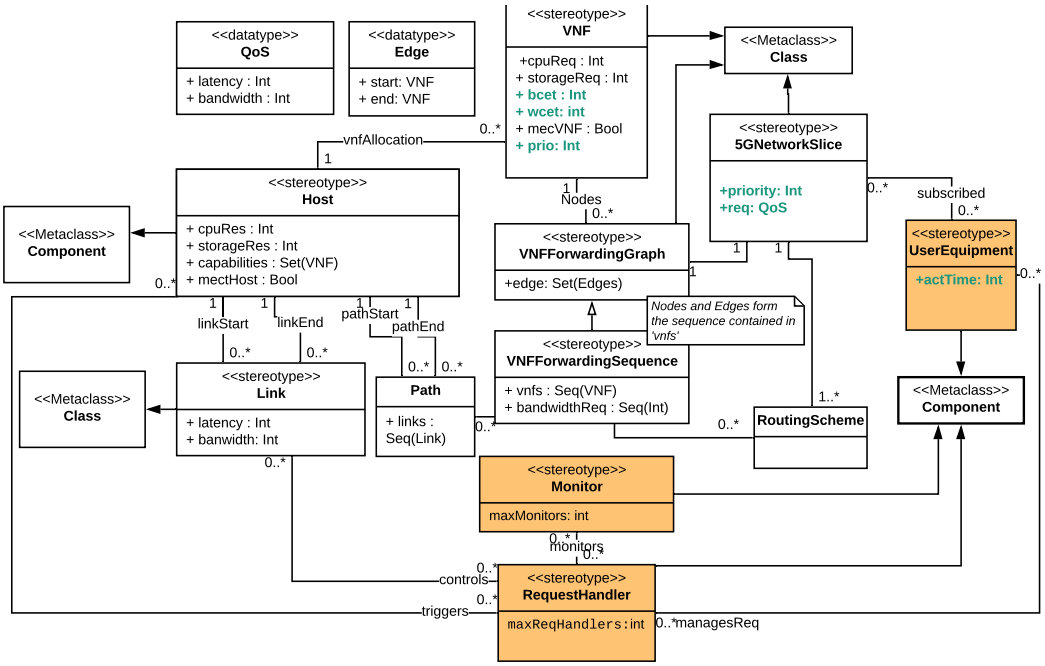

Figure 4: The UML5G-SO Profile (Extended Version)

stereotypes as follows: (i) *UserEquipment*, extending *UML Metaclass::Component*, models 5G user equipment (UE). The stereotype has an attribute to specify its activation time (assumed periodic); (ii) *Request Handler*, extending *UML Metaclass::Component*, takes care of parallel UE requests and allows VNFs to be routed according to their graph $VNFForwardingSequence$; (iii) *Monitor*, extending *UML Metaclass::Class*, monitors various served requests, and is used to establish if the latter meet their requirements or not. We also replace the execution time attribute of the previous VNF stereotype with two attributes specifying best-case (BCET) and worst-case execution time (WCET), respectively.

5.2 Case-study Modeling with Class Diagrams

The class diagram description of our case study is shown in Fig 5. We apply the stereotypes and add attributes and functions (as highlighted in Fig. 5), accordingly. We do not add these at the profile level as we acknowledge that our view of defining the system behavior is not the sole way of describing it.

We model two categories of 5G UE, namely, a $5GCamera$, and a *MobilePhone*, which access the *HospitalSlice* and $VideoSlice$, respectively. The user equipment classes contain the attributes *ueId*, *sId* and *maxReq*, which are integer values that allow to specify the UE's id, the id of the slice it accesses, and the maximum number of requests that limits the usage of the slice by the equipment. The user equipment is also associated with an action of generating the slice request event, *evSliceReq*, each time the UE gets activated periodically, with a period specified by its activation time $actTime$. Similarly, the $VideoSlice$ and *HospitalSlice* classes are defined by applying the $5GNetworkSlice$ stereotype and adding an attribute for the id. We also apply stereotypes on *ReqControl*, *MonitorReq* and VM. The *ReqControl* class is supplemented with an attribute for its id, and a set of functions: *initialise*(), *queueVNF*(), *consumeBW*(), *releaseBW*(), and *calcLDelay*(). When a UE requests a slice, the *initialise*() function initializes its $VNFFSEq$ and its routing scheme (Note: we assume that all VNFs are allocated). The *ReqControl* also takes care of queuing the respective VNFs to their hosts via the *queueVNF*() function. In order to consider the VNF dependencies arising from the respective $VNFFSeq$ of a slice, we model the queue function such that only the VNFs that are ready to be executed at a point in time get queued at the corresponding host. The functions *consumeBW*(), *releaseBW*(), *calcLDelay*() consume the link bandwidth while routing through it, release it after its usage, and calculate the respective delays in routing across the links, respectively. The VM class has one attribute (id) and two functions, *scheduleVNF*() and *dequeueVNF*(). The former function encodes the scheduling algorithm, and *dequeueVNF*() is responsible for dequeuing the VNF that is next to be executed. Once the class diagram description of our system is formulated, we encode the behavior of each active class by using a restricted form of UML statecharts. The classes that possess behavior are marked with a circle in Fig. 5.

5.3 Restricted Statechart-based Behavioral Description of our 5G-SO System

In this section, we discuss the behavioral description of our system using UML statecharts restricted to fit our needs. We define the restricted form of statecharts, as follows:

Definition 5.1. A *restricted statechart* (RSC) is a UML statechart obeying the following restrictions:

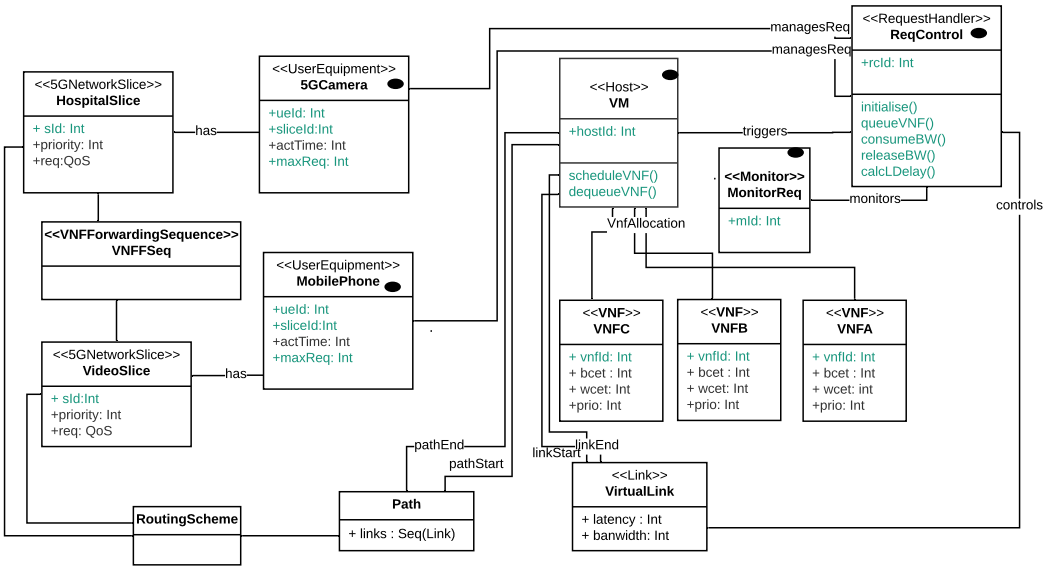

Figure 5: Class diagram representation of our case study (It inherits all relations and multiplicities in the profile)

- All states are simple (without any hierarchies), and without any associated execution history.
- The states can either be the usual simple states of UML statecharts, hereby called "active" states, or pseudostates that represent the initial and final states, only.
- The transitions follow the usual UML syntax, $Ev(parameters)[G]/A$, where $Ev(parameters)$ are either UML call events or time events, (Section 3), G are Boolean conditions evaluated over system variables, and A include variable assignments and other user-defined functions.

The transitions are triggered via call or time events, or as soon as the guards evaluate to true. Moreover, the transitions from psuedostates to active states or vice-versa are considered instantaneous if there are no specific events or guards triggering them. Different statecharts synchronize via unicast or broadcast synchronizations. The synchronization is defined via a parallel composition of the independent restricted statecharts. In addition, the statecharts follow the run-to-completion execution semantics [6], that is, a statechart completes processing each event before it can start processing the next one. The statecharts in Fig. 6 are examples of restricted statecharts, as they obey all restrictions in Def. 5.1 [1]

5.3.1 Parallel Composition of RSC(para). A 5G system comprises a number of interacting components, each behaviorally defined by an RSCpattern. Hence, to evaluate how a system is executed, we must compose in parallel the involved RSCpatterns. Given a system consisting of n RSC(para), we denote their parallel composition by:

$$RSC_1(para1)||RSC_2(para2)||\ldots||RSC_n(paran)$$

[1]The restricted statecharts in Fig. 6, namely UE, RC, VM, MO, correspond to the active classes UserEquipment, ReqControl, VM, MonitorReq, respectively.

The synchronization between the various statecharts is modeled via the call events of the form $EvName(parameter)$ between a generating and a triggered component. A component can take a transition generating an event if and only if there is another component that can trigger a transition by the same event.

Example 5.2. The UML statechart-based behavioral model of our 5G-SOS is defined as the parallel composition of the component $RSC(para)$ of our system, and is represented as follows:

$$5GSOS_{UML} = RSC_{UE1}||\ldots||RSC_{UEn}||RSC_{M01}||\ldots||RSC_{M0n}||RSC_{RC1}||\ldots||RSC_{RCn}||RSC_{VM1}||\ldots||RSC_{VMn} \quad (1)$$

where $RSC_{UE1}...RSC_{UEn}$ are the respective instantiations of $RSC_{UE}(para)$ for $UE1...UEn$ in the system. Similarly, we define instantiations of $RSC_{MO}(para)$, $RSC_{RC}(para)$ and $RSC_{VM}(para)$, representing the UML statechart patterns defined for the active classes corresponding to UserEquipment, MonitorReq, ReqControl, and VirtualMachine, respectively.

An example of synchronization is shown in Fig. 6, where the RSC_{UE} generates the call event $evSliceReq(sliceId)$ that is unicast to RSC_{MO} that synchronizes with RSC_{UE} via parameter $sliceId$.

6 FORMAL SEMANTICS OF RSC PATTERNS

In order to formally verify properties of a 5G-SO system, we need to assign formal semantics to the RSC model of component behavior, corresponding to RSC informal semantics. We define the formal semantics in terms of TA that behave according to the timed transition semantics overviewed in Sec. 3.2. For each restricted statechart pattern (see Sec. **??**), we have a corresponding TA pattern. The resulting semantics of the system is defined as the network of all TA obtained by instantiating the corresponding TA patterns.

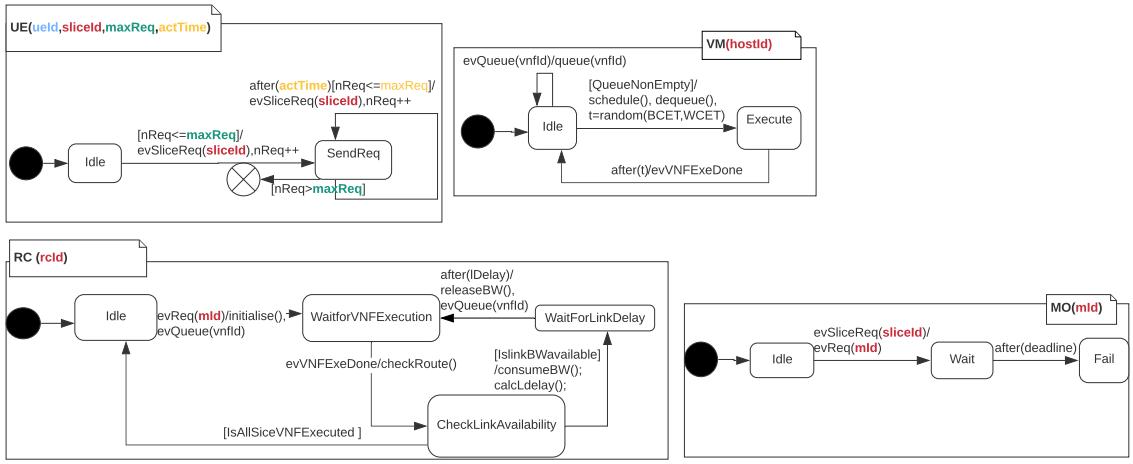

Figure 6: Restricted statechart representation of the case study specific behavior

Function	Purpose
initialise()	Initializes the slice VNFSeq, VNF allocation and routing, upon receiving a particular slice request from the UE
queue(VNFId)	Queues the VNF that are ready to be executed into the message queue of the VM where is it allocated
schedule()	Sorts the VNFs in the queue according to priorities of the VNFs such that the head of the queue has the highest priority
dequeue()	Removes the vnf that is executed from the message queue
calcLdelay()	Calculates the sum of delays over all the links involved a particular path between consecutive vnfs in VNFSeq
consumeBW()	The required bandwidth of a slice is consumed from the available link bandwidth
releaseBW()	Once the slice has finished utilizing a link, the respective link bandwidth is released

Table 1: User-defined functions

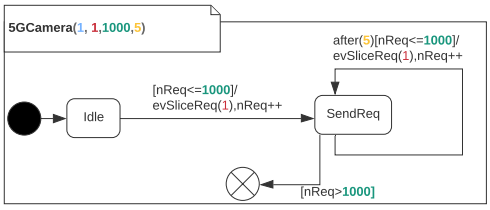

Figure 7: Example of an instantiated RSC_{UE}

UPPAAL TA pattern. Using the definition of $RSC(para)$ in Sec. **??**, and the definition of TA (see Sec. 3.2), we define a semantic encoding of the $RSC(para)$ components, respectively, in terms of $TA(para)$. Like $RSC(para)$, a $TA(para)$ is also defined as a reusable TA structure, for reccurring behavior, as follows:

$$TA(para) = \langle L_p, l_{0p}, V_p, C_p, A_p, E_p, I_p \rangle : para,$$

where:

- $para$ refers to the list of parameters that get instantiated with values when the pattern is used,
- $L_p = \bigcup_{i=1}^{n} \{La_i\} \cup \bigcup_{i=1}^{n} \{Lc_i\} \cup \bigcup_{i=1}^{n} \{Le_i\}$ where La_i correspond to Sa in $RSC(para)$, Lc_i is the set of committed locations introduced to handle the simultaneous synchronizations involving the trigger (ev_t) and effect actions (ev_g) occurring in a transition, Le_i is the set of error locations introduced to capture the errors of message queue being full, not enough request controllers or monitors to handle the request, and deadline violation,
- $l_{0p} = Idle$,
- V_p is the set of variables defined in the corresponding $RSC(para)$, and other local variables that are used to model the parameter passing (by making a local copy of the parameter passed), and other variables if needed to define the error conditions that lead to Le_i,
- C_p is the set of clock variables that measure the time elapsed for the corresponding Ev_t,
- A_p is defined as $A_{sync} \cup A_{ass} \cup A_{udf}$, where A_{sync} corresponds to the synchronizing events in Ev_c and other urgent synchronizations ($execute?$) defined to trigger a transition as soon as the guard evaluates to true, A_{ass} refers to the actions involving assignment of variables and updates of clocks, A_{udf} is the set of user-defined functions in $RSC(para)$,
- $E_p = E_e \cup E_a$, where E_e refers to the set of edges defined by $\rightarrow$ in $RSC(para)$, populated with L in $RSC(para)$ and other guards defined over the clock variables along the transitions

enabled with Ev_t; E_a are the additional edges defined to connect (La_i and Lc_i) [2] and (La_i and Le_i) [3],
- I_p is defined over the La_i that generates Ev_t.

As an example, we present the TA pattern, $TA_{UE}(para)$, depicted in Fig. 8, which corresponds to $RSC_{UE}(para)$. $TA_{UE}(para)$ is defined as follows:

$$TA_{UE}(ueId, sId, maxReq, actT) = \langle L_{ue}, l0_{ue}, V_{ue}, C_{ue}, A_{ue}, E_{ue}, I_{ue} \rangle : (ueId, sId, maxReq, actT),$$

where:

- $L_{ue} = \{Idle, Start, Select, NoFreeMonitor, NoFreeContr\}$,
- $l0_{ue} = Idle$,
- $V_{ue} = \{ueId, sId, maxReq, actT, nReq, reqSId, avContrs, avMonitors\}$, where $ueId$ represents the id of the UE, and sId represents the id of the slice it requests, $maxReq$ is the maximum number of slice requests that a UE can make, $nReq$ keeps track of the number of UE requests, $ReqSId$ is a variable that copies the sId to reflect the passing of $sliceId$ parameter in RSC_{UE}, $avContrs$, and $avMonitors$, to capture errors if there are not enough controllers or monitors to handle the requests, respectively,
- $C_{ue} = \{x\}$ is the clock that models the activation time,
- $A_{ue} = \{req?, execute?\} \cup \{reqSId = sId, nReq++, x = 0\}$, where A comprises the set of synchronization channels associated with generating a slice request ($req!$), and synchronizing over an urgent channel, ($execute?$) the corresponding variable assignments, and reset actions on clock x,
- $E_{ue} = \{Idle \xrightarrow{execute?, nReq<=maxReq, x=0} Start, Start \xrightarrow{x==actT} Select, Select \xrightarrow{avContrs==0} NoFreeContr; Select \xrightarrow{avMonitors==0} NoFreeMonitor; Select \xrightarrow{req!, (avContrs>0 \&\& avMonitors>0), nReq++, reqSId=sId, x=0} Idle\}$,
- $I_{ue} : Start \rightarrow (x <= actT)$

Following a similar approach, we generate $TA_{VM}(para)$, $TA_{RC}(para)$, and $TA_{MO}(para)$ patterns corresponding to their RSC counterparts, respectively.

The instantiation of $TA(para)$ assigns the parameters in **para** with actual values. Using "$p = v$" to denote the assignment of parameter p with value v, we define the instantiated pattern as:

$$TAI_i(para) ::= \langle L_{pi}, L_{0pi}, V_{pi}, C_{pi}, A_{pi}, E_{pi}, I_{pi} \rangle : (p_1 = v_1, p_2 = v_2, \ldots) \quad (2)$$

Example 6.1. Consider the TA_{UE} pattern, introduced above. An example of a parameter assignment is:

$$para = (ueID = 1, sId = 1, maxReq = 1000, actT = 5)$$

Parallel Composition of TA(para). Given a system consisting of n $TA(para)$, that is, $TA_1(para1), TA_2(para2), \ldots, TAn(paran)$, their parallel composition is denoted as:

$$NTA = TA_1(para1)||TA_2(para2)||...||TA_n(paran)$$

[2]If both the trigger and effect actions occur simultaneously in a transition, the latter will be transformed to two edges with a committed location in between, in which the former is synchronized with the trigger action and the latter is synchronized with the effect action.

[3]The edge connecting La with Le is decorated with guards that evaluate to true when meeting the error conditions.

Example 6.2. Assuming that we have n instantiations of each TA pattern in our model, our $5G - SOS_{NTA}$ is defined as a network of timed automata composed via parallel composition of its constituent TA models and represented as:

$$5G - SOS_{NTA} = TA_{UE1}|| \ldots ||TA_{UEn}||TA_{MO1}|| \ldots ||TA_{MOn}|| TA_{RC1}|| \ldots ||TA_{RCn}||TA_{VM1}|| \ldots ||TA_{VMn} \quad (3)$$

7 EXPERIMENTAL EVALUATION

In this section, we present an experimental evaluation of our profile. We begin by describing the tool support developed to allow for the automated verification of the requirements presented in Sec. 2 over the instantiated systems of our case study (in the form of object diagrams), followed by a short summary of our evaluation.

7.1 G^5

As a proof-of-concept of how employing our profile allows for automatic verification of object diagrams, we provide a tool, called G^5, which facilitates verifying the assertion that all deadlines are met in a specific 5G-SO system. The tool is implemented in Python and takes as input an object diagram, specified in the `soil`-format (of the USE tool [2]) and verifies automatically that for all possible scenarios all deadline requirements of slices are met. The overall workflow is described in Algorithm 1.

Input : Object diagram of 5G system
Output: Success if all deadlines are met
Instantiate TA template for each active object in system;
Create UPPAAL model with all templates;
Generate queries for UPPAAL model;
Use UPPAAL to check queries;
if *ErrorQueueFull state can be reached* **then**
  Fail with "Queues too short"
else if *NoFreeContr state can be reached* **then**
  Fail with "Not Enough Controllers"
else if *Deadline query failed* **then**
  Fail with "Deadline violated"
else
  Success

Algorithm 1: Workflow of the G^5 tool

7.2 Verification Results

We exemplify the verification approach introduced previously, for our case study described in Sec. 2, and verify whether application requirements shown in Sec. 2 are met. We begin by instantiating our TA patterns to model our case study. To model the user equipment, we instantiate the $TA_{UE}(para)$ pattern to $TA_{UE}(1, 1, 5, 5)$, $TA_{UE}(2, 1, 10, 3)$, $TA_{UE}(3, 1, 10, 100)$, $TA_{UE}(4, 2, 3, 2)$ and $TA_{UE}(5, 2, 5, 100)$, of which the first three correspond to instances of $5GCamera$, and the rest to $5GMobile$. We also instantiate the $TA_{VM}(para)$ pattern to simulate our overlay network with four VMs, namely $TA_{VM}(1)$, $TA_{VM}(2)$, $TA_{VM}(3)$, $TA_{VM}(4)$. In order to verify the model, we assume that the network has enough request controllers and monitors to handle the requests from independent

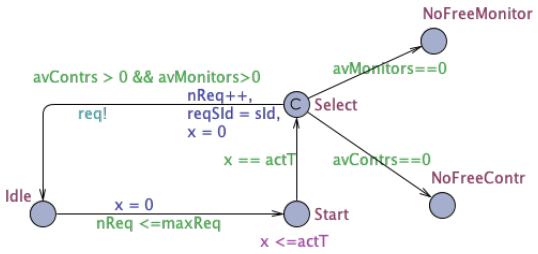

Figure 8: UPPAAL TA pattern for UE: TA_{UE}

Req.	Query	Result	Time (s)	Mem. (kB)
R1	$E\Diamond RC1.rq$	sat	0.001	7288
	$RC1.rq \rightsquigarrow$ $RC1.rqComplete$	sat	14094	3761180
R2	$A\Box$ *not MO1.Fail*	sat	12499	7920432
R3	$E\Diamond RC4.rq$	sat	0.001	7288
	$E\Diamond RC4.rqComplete$	sat	0.408	29388

Table 2: UPPAAL Verification Results

UE. However, having many such controllers and monitors is computationally expensive, hence we consider only five instances of each, hoping that they suffice.

We verify the generated model against the formalized requirements, using the UPPAAL model checker version 4.1.19 [4]. The verification PC has an Intel Core i7, 2,6 GHz processor and 16 GB 2400 MHz DDR4 memory. The verification results are tabulated in Table 2. Note that the bandwidth requirement is automatically met upon satisfaction of the latency requirement, since we construct the models in such a way that a UE can complete its request only if the necessary bandwidth is available. We also check if all slice requests are eventually served, that is, query R3 (corresponding to Req3 in Sec. 2). We also add error locations to capture queues being full, and insufficient request controllers or monitors available (e.g., see Fig. 8). In case the verification fails, the mechanism allows for detecting if the model has reached any of the error locations.

From Table 2, one can see that the time taken for exhaustive verification of certain queries is not promising, hence we acknowledge that exhaustive model checking may not always be the best solution at hand to verify big complex systems like a 5G-SOS, as it suffers from state-space explosion. However, if one is able to model and verify some critical part of the network and functionalities, it provides guarantees over all possible system behaviors.

8 RELATED WORK

Substantial work within the field of 5G service orchestration is aimed at providing optimal VNF placement algorithms and routing schemes [1, 3, 7, 16]. There is also interesting work that looks into scheduling of VNFs [4], and slice chain reconfiguration [14]. However, not much effort has been invested in modeling and formal analysis of such systems, which in turn would verify if a given VNF placement, resource allocation, and routing meets the application requirements. Nevertheless, there exists interesting work that considers the description of VNFs and VNF chaining in isolation, to analyze if application requirements are met, for instance, the Gym framework [21] and the work by Peuster and Karl [19]. In contrast to our work, these approaches model VNFs and their chaining at a low level, without considering a system perspective, or 5G-specific scenarios. Spinoso et al. [22] employ SMT solvers (e.g., Z3), to verify VNFs and VNF chains against safety and reachability properties. Although the approach is promising, the framework is strictly formal, lacking the bridge to industrially-accepted modeling languages such as UML. In addition, the authors do not consider the service-orchestration problem and the QoS requirements that we study in this paper. In another interesting work [15], Luque-Schempp et al. investigate the use of various formal methods in the context of a 5G network, focusing on Software Defined Networking and Network Function Virtualization modeling and verification via selected formal tools like theorem provers, model checkers, and SMT solvers, at different level of network abstraction, without considering 5G service orchestration. Unlike our approach, the framework does not employ modeling techniques other than those of the formal tools, which makes it less appealing to practitioners.

The use of UML to model 5G service orchestration is not investigated much in the literature. One recent work [18] models 5G network slices, namely, resource driven, service driven, deployment driven, using different UML diagrams. However the modeling is not backed by formal analysis like the one presented in this paper.

9 CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a framework to model and analyze dynamic 5G service orchestration systems. Our solution combines the features of user-friendly UML modeling with formal analysis using the UPPAAL model checker, and provides one with automated support to model and formally verify the structure and behavior of dynamic 5G SOS systems. Using our tool support requires knowledge of UML as well as the UML5G profile, but *no experience with timed automata or model checking*. This shows the power of complex automatic reasoning tools when provided to a UML user.

In our current model, we have considered only system dynamic behavior arising from simultaneous user requests, VNF sharing, variable link and server utilization, etc. In the future, we would like to consider other factors like host failures, which entails VNF reallocation or rerouting of network traffic through an alternative path. Another interesting direction for future work is to introduce an automatic translation from (any) restricted state chart to a timed automaton, allowing the specification of behavior to be changed at the UML level.

ACKNOWLEDGMENTS

This work is supported by the EU Celtic Plus/Vinnova project, Health5G - Future eHealth powered by 5G, which is gratefully acknowledged.

REFERENCES

[1] [n.d.]. D4.1. Definition of service orchestration and federation algorithms, service monitoring algorithms. http://5g-transformer.eu/index.php/deliverables/. Accessed: 2020-07-24.
[2] [n.d.]. USE: UML-based Specification Environment. https://sourceforge.net/projects/useocl/. Accessed: 2020-07-24.
[3] Satyam Agarwal, Francesco Malandrino, Carla-Fabiana Chiasserini, and Swades De. 2018. Joint VNF placement and CPU allocation in 5G. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 1943–1951.

[4] https://www.it.uu.se/research/group/darts/uppaal/download.shtml

[4] Hyame Assem Alameddine, Long Qu, and Chadi Assi. 2017. Scheduling service function chains for ultra-low latency network services. In *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE, 1–9.

[5] Rajeev Alur, Costas Courcoubetis, and David Dill. 1990. Model-checking for real-time systems. In *Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium*. IEEE, 414–425.

[6] Egon Börger, Alessandra Cavarra, and Elvinia Riccobene. 2000. Modeling the dynamics of UML state machines. In *International Workshop on Abstract State Machines*. Springer, 223–241.

[7] Vinod Kumar Choyi, Ayman Abdel-Hamid, Yogendra Shah, Samir Ferdi, and Alec Brusilovsky. 2016. Network slice selection, assignment and routing within 5G networks. In *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 1–7.

[8] Bruce Powel Douglass. 2004. *Real time UML: advances in the UML for real-time systems*. Addison-Wesley Professional.

[9] Sébastien Gérard and Bran Selic. 2008. The uml–marte standardized profile. *IFAC Proceedings Volumes* 41, 2 (2008), 6909–6913.

[10] Ashalatha Kunnappilly, Peter Backeman, and Cristina Seceleanu. 2020. UML-based Modeling and Analysis of 5G Service Orchestration. (2020).

[11] Kim G Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a nutshell. *International journal on software tools for technology transfer* 1, 1-2 (1997), 134–152.

[12] Aris Leivadeas, George Kesidis, Mohamed Ibnkahla, and Ioannis Lambadaris. 2019. VNF Placement Optimization at the Edge and Cloud. *Future Internet* 11, 3 (2019). https://doi.org/10.3390/fi11030069

[13] X. Li, M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo, and R. Jain. 2017. Network Slicing for 5G: Challenges and Opportunities. *IEEE Internet Computing* 21, 5 (2017), 20–27. https://doi.org/10.1109/MIC.2017.3481355

[14] Yicen Liu, Hao Lu, Xi Li, and Donghao Zhao. 2019. An approach for service function chain reconfiguration in network function virtualization architectures. *IEEE Access* 7 (2019), 147224–147237.

[15] Francisco Luque-Schempp, Pedro Merino-Gómez, Laura Panizo, et al. 2019. How formal methods can contribute to 5g networks. In *From Software Engineering to Formal Methods and Tools, and Back*. Springer, 548–571.

[16] Tahira Mahboob, Young Rok Jung, and Min Young Chung. 2020. Dynamic vnf placement to manage user traffic flow in software-defined wireless networks. *Journal of Network and Systems Management* (2020), 1–21.

[17] Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. 2019. A framework for verification-oriented user-friendly network function modeling. *IEEE Access* 7 (2019), 99349–99359.

[18] A. Papageorgiou, A. Fernández-Fernández, S. Siddiqui, and G. Carrozzo. 2020. On 5G network slice modelling: Service-, resource-, or deployment-driven? *Computer Communications* 149 (2020), 232–240.

[19] Manuel Peuster and Holger Karl. 2016. Understand your chains: Towards performance profile-based network service management. In *2016 Fifth European Workshop on Software-Defined Networks (EWSDN)*. IEEE, 7–12.

[20] Marcelo A Ramos, Paulo C Masiero, Rosangela AD Penteado, and Rosana TV Braga. 2015. Extending statecharts to model system interactions. *Journal of Software Engineering Research and Development* 3, 1 (2015), 1–25.

[21] Raphael Vicente Rosa, Claudio Bertoldo, and Christian Esteve Rothenberg. 2017. Take your vnf to the gym: A testing framework for automated nfv performance benchmarking. *IEEE Communications Magazine* 55, 9 (2017), 110–117.

[22] Serena Spinoso, Matteo Virgilio, Wolfgang John, Antonio Manzalini, Guido Marchetto, and Riccardo Sisto. 2015. Formal verification of virtual network function graphs in an sp-devops context. In *European Conference on Service-Oriented and Cloud Computing*. Springer, 253–262.