Computer Science and Information Systems 20(1):329–357 https://doi.org/10.2298/CSIS211001059B

Supporting 5G Service Orchestration with Formal Verification*

Peter Backeman[1], Ashalatha Kunnappilly[2], and Cristina Seceleanu[1]

[1] Mälardalen University,
Västerås, Sweden
{peter.backeman,cristina.seceleanu}@mdu.se
[2] Alstom,
Västerås, Sweden
ashalatha.kunnappilly@alstomgroup.com

Abstract. The 5G communication technology has the ability to create logical networks, called network slices, which are specifically carved to serve particular application domains. Due to the mix of different application criticality, it becomes crucial to verify if the applications' service level agreements are met. In this paper, we propose a novel framework for modeling and verifying 5G orchestration, considering simultaneous access and admission of new requests to slices as well as virtual network function scheduling and routing. By combining modeling in user-friendly UML, with UPPAAL model checking and satisfiability-modulo-theories-based model finding, our framework supports both modeling and formal verification of service orchestration. We demonstrate our approach on a e-health case study showing how a user, with no knowledge of formal methods, can model a system in UML and verify that the application meets its requirements.

Keywords: 5G, model checking, SMT, UML

1. Introduction

The fifth generation of wireless technology, 5G, has the potential to support a variety of applications with different requirements, be it 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 [15]. 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 latency. Adding to the complexity, VNFs can be shared between slices that are requested simultaneously by various 5G user equipment (UE).

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 network resources, one needs to ensure that all slices facilitate meeting the requirements of considered applications.

* This is an extension of previous work published at ECBS 2021 [12].

Although much research has been devoted to solving the 5G service orchestration problem by providing optimal VNF allocation and routing schemes [19,14], there is a lack of endeavors that provide modeling and formal verification frameworks that can analyze such schemes to provide guarantees of the intended system behavior. In this paper, we propose such a modeling and formal analysis framework that combines user-friendly UML-based modeling [8] with mathematical assurance via exhaustive model checking in UPPAAL [13] and model finding using Z3 [21]. This work builds on our previous results [11], the UML5G-SO framework, which allows for modeling and analysis or VNF allocation and routing, assuming static worst-case scenarios. We augment the UML5G-SO framework to support dynamic system behavior stemming from slice requests from UE, 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 Sec. 4.4), (iii) defining pattern-based timed automata semantics for restricted statechart patterns, enabling model-checking UML5G-SO behaviors with UPPAAL [13], (iv) defining an alternative semantics in first-order logic with linear arithmetic for the restricted statechart patterns, as well as (v) implementing tool support for the automatic generation of UPPAAL and SMT models from UML diagrams. To demonstrate our approach, we consider a case study of simultaneous access of shared network resources by two applications of different criticality. This paper is an extension of our earlier work [12], and in this extended journal version we have added a formalization for semantics using first-order logic with linear arithmetic, and support for generating SMT models. While we show that the scalability of SMT models surpass that of model checking, the possibility of simulating traces step-by-step is lost and queries must be expressed in first-order logic (in contrast to TCTL queries). Hence, the two semantics facilitate two complementary approaches for verification.

The goal is to create a framework in *which a user with no experience in formal methods can use to formally verify properties of a 5G service orchestration system*. The usability is our prime concern, and we wish to provide an interface which relates to the service orchestration-problem and hides as many verification details as possible.

The rest of the paper is organized as follows. Sec. 2 details the problem statement, In Sec. 3, we give an overview of UML 2.0 modeling, timed automata, the UPPAAL model checker, first-order logic, and the Z3 tool. In Sec. 4, we present our extended UML5G-SO profile in UML 2.0 allowing UML modeling of dynamic service orchestration in 5G systems. Sec. 5 presents the formal semantics of the UML model in terms of UPPAAL timed automata, while Sec. 6 presents alternative semantics in first-order logic. Thereafter, in Sec. 7, we present our methodology and the G^5 tool that allows the automated verification of system requirements, followed in Sec. 8 by a case study with experimental verification results, continued by a brief discussion of the gained insights. We compare our contribution to related work in Sec. 9, before presenting the concluding remarks and directions of future work, in Sec. 10.

2. Problem Description

This paper aims at providing a modeling and formal verification framework for dynamic 5G service orchestration. Concretely, the framework allows a 5G engineer to model an

existing service orchestration configuration (that is, VNF allocation and routing) of a network slice, and 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 hosts executes VNFs according to a given scheduling policy, with an execution time within its best- and worst-case time bounds.
- *Dynamic link utilization:* Based on the execution of the VNFs, we model the utilization of each link's bandwidth, to ensure that it is never over-utilized.

Consider an overlay network consisting of virtual machines (hosts). We assume that this network is powered by 5G, which supports end-to-end network slices. A slice consists of a number of virtual network functions (VNF), generally interconnected via a VNF Forwarding Graph, in this paper assumed to be a VNF Forwarding *Sequence*. A virtual network function is defined as a software implementation of a network function, which can be easily deployed on virtual resources [14]. We also assume that the hosts communicate via virtual links, which incurs overheads in terms of bandwidth capacity and latency. For the overlay network to serve various applications, the network slices' VNFs need to: (i) be allocated on hosts, respecting processing, memory, and storage capabilities, and (ii) be routed such that the respective VNF chaining is achieved. In our previous work [11], we have proposed a UML profile called UML5G-SO, and associated static OCL-based analysis of 5G service orchestration solutions, for checking if a given VNF allocation and routing of a slice meets the application's Quality-of-Service (QoS) requirements. However, our analysis considered only static worst-case scenarios, assuming that the system is serving a maximum number of user requests under a maximum load. This is unrealistic if one considers varying network load using hosts and links in a dynamic manner.

We assume that each slice has a certain allocation and routing defined 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 with model checking and with SMT model finding.

2.1. Running Example

To illustrate our framework we will use a small *running example* in this paper. We consider a small use case where we only have one user equipment subscribing to one slice with a latency requirement of 50 ms. The slice consists of two VNFs, each allocated to a separate host, connected by a single link. The use case is illustrated in Fig. 1, where the single slice S_1 consists of V_1 and V_2 (both of the same type). The requirement is to ensure that executing VNF V_1 on H_1, traversing the link L_1 and executing VNF V_2 on H_2 in total takes less time than the slice deadline of 50 ms.

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 *satisfiability modulo theories* (SMT).

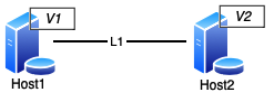

Fig. 1. Illustration of the running example

3.1. UML Diagrams

The Unified Modeling Language (UML) [8] is a modeling language that helps designers to express structural and behavioral artifacts of complex systems. In order to capture the essentials of a 5G-SO system, we specify its structure and behavior, by using our UML5G-SO profile. A *profile diagram* is used to 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 [9,8].

We use UML *class diagrams* and *object diagrams* to represent a system's structure, and UML *statecharts* to model a system's behavior. While the former are quite straightforward, we provide a brief explanation of statecharts, as they are central to this work. 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* (Ev), and the response actions (A) become the effects on the transitions. The *guard* (G) 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 $Ev(parameters)[G]/A$. We chose UML statecharts for their effectiveness of capturing the behavior of individual classes (system's components).

In our work, we consider only two kinds of UML events, *time events* and *call events* [8]. We define a time event via the keyword "after", followed by an expression encoding a time value. We model call events as synchronization events that are unicast to other statecharts. The unicast communication offers handshake synchronization between two statecharts, and is blocking, i.e., the synchronization takes place only if both sender and receiver are ready to traverse their edges [24]. In addition, we consider that statecharts can be composed in parallel, defining their interactions via call events [24].

Example 1. Fig .2 shows an example of two statecharts that synchronize when executing in parallel. The initial one is the *controller* 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 a call event (action), $sendTrigger(vnfId)$, which is unicast to the *host*. In state Sa, the host 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 [13], 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

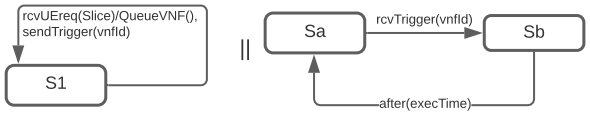

Fig. 2. Parallel Composition of UML Statecharts

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 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 $u' \vDash I(l')$.

The UPPAAL model checker provides exhaustive model-checking of TA models like the ones overviewed. 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 (via $c!$ and $c?$). 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) [4], and checked by the UPPAAL model checker. UPPAAL supports verification of liveness and safety properties [13]. The queries that we verify in this paper are of the form: **Invariance**, $A\Box p$, stating that p should be true in all reachable states for all paths.

3.3. Satisfiability Modulo Theories and Z3

In this paper, we use a standard approach towards first-order logic using Boolean operators ($\wedge, \vee, \Rightarrow$) as well as quantifiers ($\forall, \exists$). A formula is *satisfiable* if there is a model (an interpretation) that makes the formula true, otherwise it is called *unsatisfiable*. In this work quantifiers can be handled by enumeration as all underlying domains are finite. For an introduction and more we refer the reader to [26].

The satisfiability modulo theories (SMT) problem is posed as: given a formula, together with one (or more) background theories, is the formula satisfiable or not? In this paper we use the Z3 SMT solver [21], which can both answer such queries modulo the theory of linear arithmetic, as well as provide concrete values of a model (which is useful when analyzing answers).

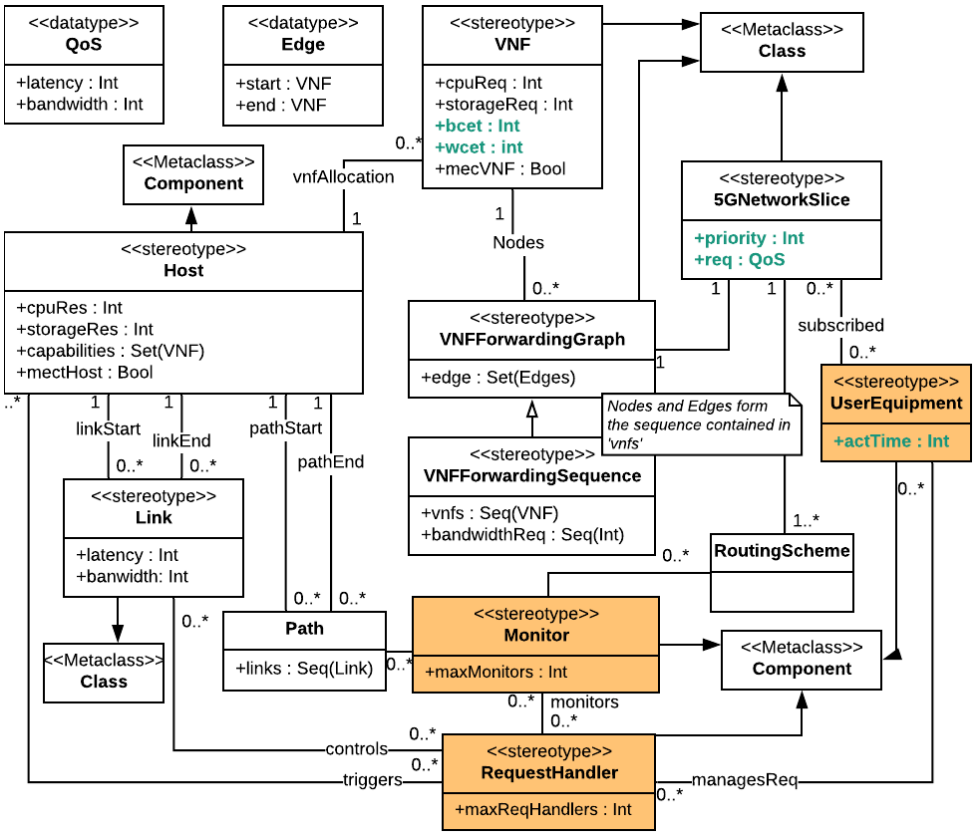

Fig. 3. The UML5G-SO Profile (Extended Version)

4. UML Modeling of Dynamic 5G Service Orchestration

In this section, we exemplify how the UML5G-SO profile can be used for modeling 5G-specific scenarios using our running example.

4.1. The Enhanced UML5G-SO Profile

The enhanced version of the UML5G-SO profile is shown in Fig. 3 (changes highlighted with colors and in bold). We define three stereotypes as follows: (i) *UserEquipment*, extending *UML Metaclass::Component*, models 5G 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 VNF forwarding sequence; (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).

4.2. Class Diagram Modelling of System Components

The class diagram is used to describe component types for a system. Using the stereotypes from the profile and adding attributes and functions, we introduce a class for each component type (e.g., a slice type, a kind of host). These are used by object diagrams to specify instances of systems, which can be analyzed using the methods in this paper.

Example 2. To illustrate, we show in Fig. 4 a class diagram for a running example. Some of the classes, $MonitorReq$ and $ReqControl$, are defined because they are required to model the full behavior of the system. The rest are introduced to model the components of the system, e.g., the *Ethernet* and *Server*).

In this example there is only one slice type, $5GCamera$ accessing $Slice$. The user equipment class is created by applying the $UserEquipment$ stereotype and contains the attributes $ueId$, sId and $maxReq$, which specify its id, the id of the slice it accesses, and the maximum number of requests the UE can perform, respectively. The user equipment generates the slice request event, $evSliceReq$, each time the UE gets activated, with a period equal to its activation time $actTime$. Similarly, the $Slice$ class 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 VNF forwarding sequence and its routing scheme (note that we assume that all VNFs are allocated). The $ReqControl$ also takes care of queuing the respective VNFs to their hosts via the $queueVNF()$ function. 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. The VM class has one attribute, id and two functions, $scheduleVNF()$ and $dequeueVNF()$, the former encoding the scheduling algorithm, and the latter being responsible for dequeueing the next VNF to be executed.

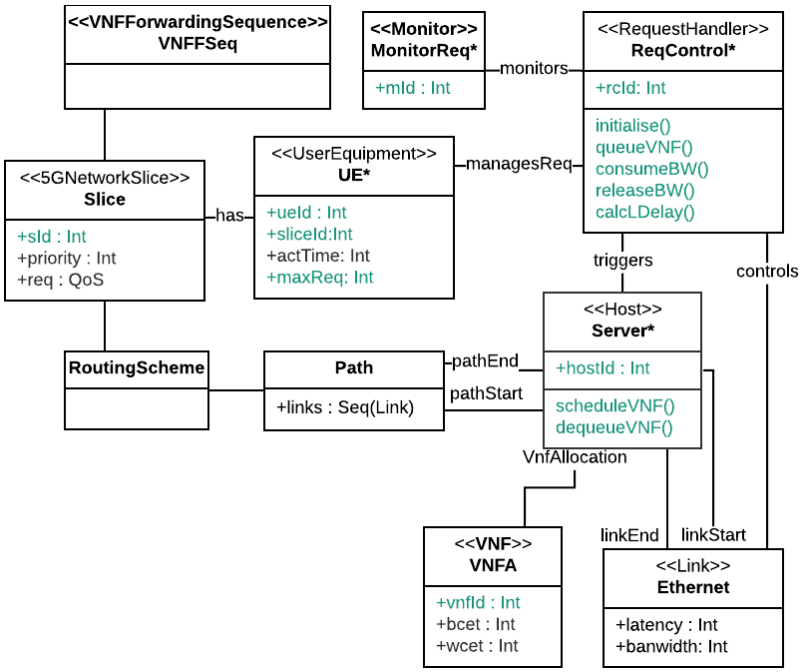

Fig. 4. Class Diagram for our running example

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. In our examples, the classes that possess behavior are marked with an asterisk, e.g., see Fig. 4.

4.3. Object Diagram Description of System Instance

Given a class diagram, describing each component in the system, it is possible to depict a system instance by means of an *object diagram*. In the object diagram, the classes are instantiated and their attributes assigned values, according to the corresponding use case.

Example 3. In Fig. 5 an object diagram shows the running example system instance.

4.4. 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 1. *A* restricted statechart *(RSC) is a UML statechart obeying the following:*

- All states are simple (without hierarchies) and without 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.

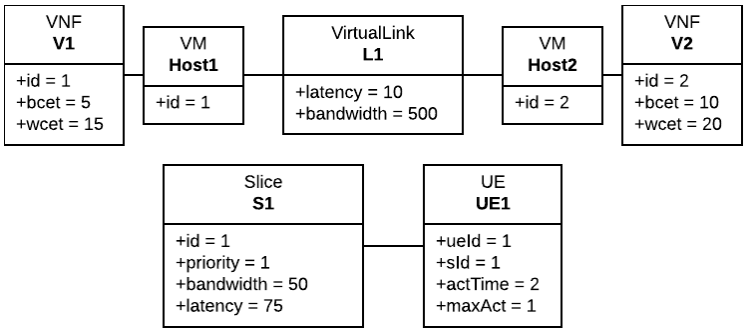

Fig. 5. Object Diagram for our running example

– The transitions follow usual UML syntax, $Ev(params)[G]/A$, where $Ev(params)$ are UML call or time events (Sec. 3), G are Boolean conditions evaluated over system variables, and A include variable assignments and other user-defined functions.

A transition $Ev(params)[G]/A$ is triggered when the event $Ev(params)$ occurs and G is true, or as soon as G is true in case of no triggering event. Moreover, the transitions from psuedostates to active states or vice-versa are considered instantaneous if there are no triggering events or guards enabling them. Different statecharts synchronize via unicast or broadcast synchronizations, defined via a parallel composition of the independent restricted statecharts. In addition, the statecharts follow the run-to-completion execution semantics [5], i.e., 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

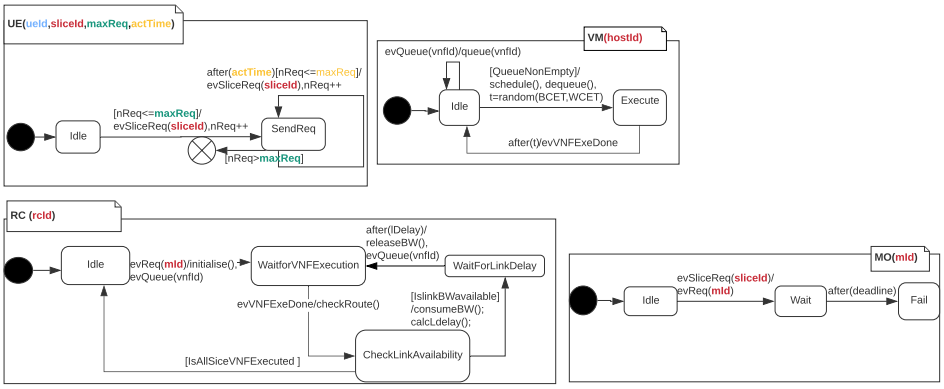

Fig. 6. Restricted statechart representation of the case-study-specific behavior

obey all restrictions in Def. 1. Note that the statecharts UE, RC, VM and MO, correspond to the active classes UserEquipment, ReqControl, VM, MonitorReq, respectively.

Pattern-Based Behavioral Representation To model the behavior of the components, we apply a pattern-based representation. We begin with an auxiliary definition.

Definition 2. *An* assignment *for a set of variables V, is a mapping from variables to values such that each variable is assigned one value. The set of possible assignments for V is denoted as $\mathcal{A}_V$.*

A *restricted statechart pattern, RSC(para)* is a reusable RSC structure used to define repetitive behaviors, and is represented as the following tuple:

$$RSC(para) = \langle S_p, s_{ip}, V_p, Ev_p, F_p, L_p, \rightarrow_p \rangle : para, \tag{1}$$

where:

- $para$ refers to the list of parameters instantiated with values when the pattern is used,
- $S_p = S_{act} \cup S_{psu} = \{s_1, .., s_n\}$, is the set of states, i.e., the set of active states (S_{act}), and the set of pseudostates (S_{psu}), that is, initial and final states,
- $s_{ip} \in S$ is the initial pseudostate,
- V_p is the set of system variables,
- $Ev_p = Ev_t \cup Ev_c$ is the set of events, where Ev_t is the set of time events and Ev_c is the set of call events. Further, $Ev_c = Ev_{trig} \cup Ev_{act}$, where Ev_{trig} is the set of events triggering the transition, and Ev_{act} is the set of generated events,
- F_p is the set of user-defined functions, where for each $f \in F_p, f : \mathcal{A}_{V_p} \mapsto \mathcal{A}_{V_p}$, which maps each variable assignment to a (possibly identical) variable assignment,
- $L_p \subseteq Ev_p \times G_p \times \mathcal{A}_{V_p} \times F_p \times Ev_p$ is the set of labels, where for each label $(ev_t, g, a, f, ev_g) \in L_p$:
  - ev_t is a *triggering* event,
  - g is a guard (i.e., G_p is the set of Boolean expressions over V_p),
  - a is an assignment,
  - f is a user-defined function,
  - ev_g is a *generated* event.
- $\rightarrow_p \subseteq S_p \times L_p \times S_p$, denoted by $s \xrightarrow{L_p}_p s', \{s, s'\} \in S_p$.

Example: In Fig. 6 we present the four patterns corresponding to the active classes in Fig. 10. We exemplify the restricted statechart $RSC_{UE}(ueId, sliceId, maxReq, actTime)$, describing the behavior of UserEquipment:

- $S_p = S_{psu} \cup S_{act}$, where $S_{psu} = \{Initial, Final\}$ and $S_{act} = \{Idle, SendReq\}$,
- $s_{ip} = Initial$,
- $V_p = \{nReq\}$, $para = \{ueId, sliceId, maxReq, actTime\}$,
- $Ev_p = \{evSliceReq(sliceId), after(actTime)\}$,
- $F_p = \{\emptyset\}$,
- $L_p = \{(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset), (\emptyset, nReq \leq maxReq, nReq{+}{+}, \emptyset, evSliceReq(sliceId))$, $(after(actTime), nReq \leq maxReq, nReq{+}{+}, \emptyset,$ $evSliceReq(sliceId)), (\emptyset, nReq > maxReq, \emptyset, \emptyset, \emptyset)\}$,
- $\rightarrow = \{Initial \rightarrow Idle, Idle \xrightarrow{\emptyset, nReq \leq maxReq, nReq{+}{+}, \emptyset, evSliceReq(sliceId)} SendReq,$ $SendReq \xrightarrow{after(actTime), nReq \leq maxReq, nReq{+}{+}, \emptyset, evSliceReq(sliceId)} SendReq,$ $SendReq \xrightarrow{\emptyset, nReq > maxReq, \emptyset, \emptyset, \emptyset} Final\}$

Once the patterns are defined, the patterns can be instantiated by assigning $p \in para$ with actual values. We denote by "$p = v$" the assignment of parameter p with value v, so the instantiated RSC(para) is:

$$RSC_i(v_1, v_2, \ldots) = \langle S_{pi}, s_{ip_i}, V_{pi}, Ev_{pi}, F_{pi}, L_{pi}, \rightarrow_{pi} \rangle : (p_1 = v_1, \ldots) \quad (2)$$

Example 4. Consider the RSC_{UE} pattern. An example of a parameter assignment is:

$$para = \{(ueID, 1), (sliceID, 1), (maxReq, 1000), (actTime, 5)\} \quad (3)$$

The resulting statechart is shown in Fig. 7.

User-Defined Functions. Some of the classes contain user-defined functions, expressing some behavioral aspect of the system. These must be formalized both for the TA model as well as the SMT model. In general, for the TA model we consider user-defined functions to be given as C-like code, as supported by UPPAAL. For the SMT model instead, the effect of executing the function must be formulated in first-order logic. A list of user-defined functions used in our model is found in Table 1.

Example 5. Consider the function $schedule()$ from our running example. Its C-like code is shown in Listing 1.1. A SMT-formalization of its effect is presented later (see Eq. 42).

Listing 1.1. Function for priority-based scheduling

```
int i, j, tmp;
for (i = 0; i<VM.MQ.length; i++) {
    for (j = 1; j<VM.MQ.length; j++) {
        if (VM.MQ[i].Prio > VM.MQ[j].Prio) {
            tmp = VM.MQ[i]
            VM.MQ[i] = VM.MQ[j]
            VM.MQ[j] = temp
        }
    }
}
```

Table 1. User-defined functions

Function	Purpose
$initialise()$	Initializes the slice VNF forwarding sequence, 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 it is 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 latencies over all the links involved a particular path between consecutive VNFs in VNF forwarding sequence.
$consumeBW()$	The required bandwidth of a slice is consumed from the available link bandwidth.
$releaseBW()$	Once a slice has finished utilizing a link, release the respective bandwidth.

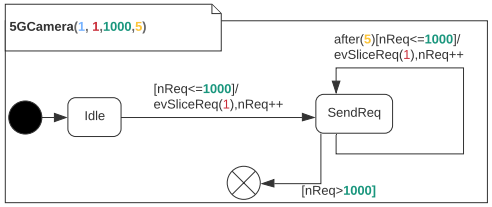

Fig. 7. Example of an instantiated RSC_{UE}

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

$$RSC_1(para_1) \,||\, RSC_2(para_2) \,||\, \cdots \,||\, RSC_n(para_n) \tag{4}$$

Example 6. The UML statechart-based behavioral model of our running example is defined as the parallel composition of each system component's $RSC(para)$:

$$RSC_{UE1} \,||\, RSC_{MO1} \,||\, RSC_{RC1} \,||\, RSC_{Host1} \,||\, RSC_{Host2} \tag{5}$$

where RSC_{UE1} is the instantiations of the pattern $RSC_{UE}(para)$ for $UE1$ 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 MonitorReq, ReqControl, and VirtualMachine, respectively.

Request Controllers and Monitors. There are two active classes, request controllers and monitors, which play a special role in the model. These are simulation artifacts that facilitate keeping track of which VNF is currently executed where, and what has to be done next. The model only requires that sufficient instances of these two classes are provided.

5. TA Semantics of RSC Patterns

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 semantics in terms of TA that behave according to timed transition semantics (see Sec. 3.2). For each restricted statechart pattern (see Sec. 4.4), we have a corresponding TA pattern. The semantics of the system is defined as the underlying timed transition system of the network of all TA obtained by instantiating the corresponding TA patterns.

5.1. UPPAAL TA pattern

Using the definition of $RSC(para)$ in Sec. 4.4, 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 reoccurring behavior, as follows:

$$TA(para) = \langle L_p, l_{0p}, V_p, C_p, A_p, E_p, I_p \rangle : para, \tag{6}$$

where:

- $para$ is a list of parameters that gets 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 S_{act} 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 errors of message queue being full, not enough request controllers/monitors to handle the request, or deadline violation,
- $l_{0p} = Idle$ is the initial location,
- 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,
- $A_p = A_{sync} \cup A_{asg} \cup A_{udf}$, 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_{asg} is the actions involving assignment of variables and clock resets, A_{udf} is the set of user-defined functions in $RSC(para)$,
- $E_p = E_e \cup E_a$, are the TA edges, where E_e refers to the set of edges defined by $\rightarrow_p$ in $RSC(para)$, decorated with L_p in $RSC(para)$ and other guards defined over the clock variables along the transitions enabled with Ev_t events; E_a are the edges defined to connect (La_i and Lc_i) [3] and (La_i and Le_i) [4],
- I_p is the set of invariants that are defined over La_i that generate Ev_t.

Example 7. 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:

$$\begin{gathered} TA_{UE}(ueId, sId, maxReq, actT) = \\ \langle L_{ue}, l0_{ue}, V_{ue}, C_{ue}, A_{ue}, E_{ue}, I_{ue} \rangle : (ueId, sId, maxReq, actT), \end{gathered} \tag{7}$$

where:

- $L_{ue} = \{Idle, Start, Select, NoFreeMonitor, NoFreeContr\}$,
- $l0_{ue} = Idle$,
- $V_{ue} = \{ueId, sId, maxReq, actT, nReq, reqSId, avContrs, avMonitors\}$, such that $ueId$ 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 no available controllers or monitors to handle the requests,

[3] 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.

[4] The edge connecting La and Le is decorated with guards that are true when meeting the error conditions.

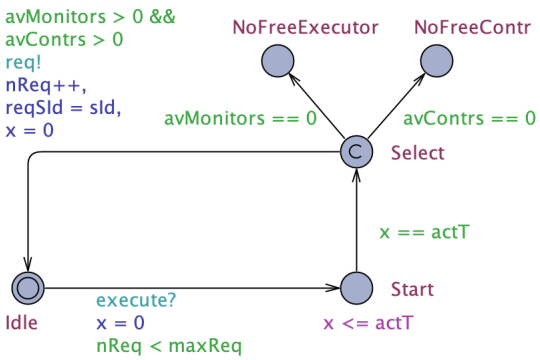

Fig. 8. UPPAAL TA pattern for UserEquipment

- $C_{ue} = \{x\}$ is the one-clock set containing 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 enforcing UE to not idle ($execute?$), the corresponding variable assignments, and reset action on clock x,
- $E_{ue} = \{Idle \xrightarrow{execute?, nReq<maxReq, x=0} Start, Start \xrightarrow{x==actT} Select,$
  $Select \xrightarrow{avContrs==0} NoFreeContr, Select \xrightarrow{avMonitors==0} NoFreeMonitor,$
  $Select \xrightarrow{req!, (avContrs>0 \&\& avMonitors>0), nReq{+}{+}, reqSId=sId, x=0} Idle\}$,
- $I_{ue} : Start \rightarrow (x \leq actT)$.

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

$$TA_i(v_1, v_2 \ldots) ::= \langle L_{pi}, l_{0pi}, V_{pi}, C_{pi}, A_{pi}, E_{pi}, I_{pi} \rangle : (p_1 = v_1, p_2 = v_2, ...) \quad (8)$$

Example 8. Consider TA_{UE} introduced above. An example parameter assignment is:

$$para = (ueID = 1, sId = 1, maxReq = 1000, actT = 5) \quad (9)$$

Parallel Composition of TA(para). Given a system consisting of n $TA(para)$, that is, $TA_1(para_1), TA_2(para_2), \ldots, TA_n(para_n)$, their parallel composition is denoted as:

$$NTA = TA_1(para_1) \parallel TA_2(para_2) \parallel \ldots \parallel TA_n(para_n) \quad (10)$$

Example 9. Consider the running example. Assuming that there are only one parallel request, the behavior is formalized as a network of timed automata:

$$TA_{UE1} \parallel TA_{MO1} \parallel TA_{RC1} \parallel TA_{Host1} \parallel TA_{Host2} \quad (11)$$

5.2. Queries

Given an instantiated UPPAAL model, we can formulate (T)CTL queries to check properties of the system. For example, the following query is satisfied if there is no scenario where a deadline is violated (i.e., checking that the monitor does not register a missed deadline):

$$A\,\Box\ not\ MO.MissedDeadline \tag{12}$$

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.

6. First-Order Semantics

In the previous section, we assigned semantics to the 5G system in the form of timed automata. As an alternative, in this section we present a formalization of semantics in first-order logic with linear arithmetic. We show how the behavior depicted in the statechart in Fig. 6 can be represented. As mentioned, in this paper we assume that we already have a sound allocation and routing; nevertheless, it is also straightforward to encode this in logic, see our previous work where a formalization was presented in OCL [11].

As a starting point, we consider a given 5G-SO system described by an object diagram. We present a set of formulas that model the intended semantics, and then we add constraints for the desired property to be checked, e.g., "deadlines are not violated".

Example 10. Consider the network given in Fig. 5, consisting of two hosts and one link. On this network, we wish to serve one slice consisting of two VNFs. All requests for slices are generated by a single user equipment.

We start with a formalization of the objects found in an object diagram.

Definition 3. *We define a 5G-SO system as the tuple:* $5GSO = (H, L, S, V, U)$*, where* H *is the set of* Hosts*,* L *is the set of* Links*,* S *is the set of* Slices*,* V *is the set of* VNFs *and* U *is the set of* User Equipment*.*

Each object has a number of properties, e.g., a slice has a *deadline* and a *VNF forwarding sequence*, which are listed in Table 2. In formulas, we use $prop(o)$ to denote the value of the prop assigned to object o. For example, $deadline(s)$ denotes the *deadline* assigned to slice s. All relevant properties and corresponding functions are shown in Table 2.

Example 11. Our running example has $H = \{H_1, H_2\}$, $L = \{L_1\}$, $S = \{S_1\}$, $V = \{V_1, V_2\}$ and $U = \{U_1\}$. Properties from the object diagram are defined accordingly, e.g., $bw(L_1) = 500$, $deadline(S_1) = 75$, $vnffseq(S_1) = [V_1, V_2]$, and $subslice(U_1) = S_1$.

We introduce two core concepts for our model: *requests* and *schedules*. A request describes, for a given slice, what VNFs needs to be executed, when and for how long. Since our model assumes an allocation of VNFs to hosts, and given best-case and worst-case execution times, we can represent a request as a sequence of host and durations.

Table 2. Properties of a 5G system

Object type	Description	Type	Function name
Link	Bandwidth capacity of the link	Integer	$capacity$
Link	Latency of the link	Integer	$latency$
Slice	Latency requirement of slice	Integer	$deadline$
Slice	Constituting sequence of VNFs	Sequence of VNF types	$vnffseq$
VNF	Best-Case Execution Time	Integer	$bcet$
VNF	Worst-Case Execution Time	Integer	$wcet$
User Equipment	Subscribed slice	Slice	$subslice$

Definition 4. *A* request R *is defined as a constraint-triple* $(R_{start}, R_{deadline}, R_{priority})$ *and a sequence of host-duration triples:*

$$R_{exec} = [(h_1, dmin_1, dmax_1), \ldots, (h_n, dmin_n, dmax_n)], \tag{13}$$

where $_{start}, R_{deadline}, R_{priority} \in \mathbb{N}$*,* $h_i \in H$ *and* $dmin_i, dmax_i \in \mathbb{Z}^+$.

The constraint-triple gives the starting time of a request ((R_{start}), its (relative) deadline ($R_{deadline}$), and its priority ($R_{priority}$). The host-duration triple consists of a host, a minimum and a maximum execution time. Intuitively, $(h_1, 5, 15)$ means the request requires host h_1 to execute between 5 and 15 time units. This information is derived from the best- and worst-case execution time of a VNF, as well as to which host it is allocated.

Example 12. In our running example, considering that the user equipment has only two activation, it generates two requests R^1 and R^2 according to S_1, which consists of VNFs $[V_1, V_2]$, and thus: $R^1_{exec} = e$, $R_{start} = 0$, $R_{deadline} = 75$ and $R_{priority} = 1$, and $R^2_{exec} = e$, $R_{start} = 50$, $R_{deadline} = 125$ and $R_{priority} = 1$, where $e = [(H_1, 5, 15), (H_2, 10, 20)]$.

Our second concept describes how a request can be fulfilled. A *schedule* is a sequence of *actions*. Each action is either a computation-based *host*-action, a communication-based *link*-action or one of the *meta*-actions initialize and complete. Each action has a time-tuple that describes when the corresponding is executed.

Definition 5. *A* schedule *for a request* R*, denoted* S_R*, is a sequence of actions, each action being one of the following:*

- $\mathrm{I}(T)$ *- This signifies the start of the request*
- $\mathrm{H}^p_h(T)$ *- This signifies the host* h *computing with priority* p
- $\mathrm{L}^w_l(T)$ *- This signifies link* l *communicating with and bandwidth of* w
- $\mathrm{C}(T)$ *- This signifies the completion of the request*

where $T = (t_0, t_B, t_E)$ *is the timing information for the action. It is interpreted as follows:* t_0 *is the first possible time the action can be performed;* t_B *is the time when the action is* actually *started, and* t_E *is the time when the action is finished. It is required that the schedule begins with an initialize-action and ends with a complete-action.*

Note that for the meta-actions all time-values will be equal, so we can write $\mathrm{I}(0,0,0)$ as $\mathrm{I}(0)$ (and similar for the completion action).

In contrast to Sec. 5 where we have described the semantics from a behavioral point of view, here we describe the possible resulting execution traces. For a particular request, we know which hosts need to execute, for how long, with what priority, as well as what links should be traversed in between with what delay. Therefore, we can for each request introduce a schedule *fulfilling* that request, i.e., it performs the necessary actions.

Example 13. The following schedule fulfills the request R^1 in Example 12:

$$[\mathrm{I}(0), \mathrm{H}_1^1(0,0,10), \mathrm{L}_1^{50}(10,10,20), \mathrm{H}_2^1(20,30,40), \mathrm{C}(40)] \tag{14}$$

The schedule is initialized at time zero, then H_1 computes for ten time units, L_1 communicates for ten time units, and finally H_2 computes for ten time units. Note that the second host-action has a delay between when it was available and when it was started.

For a set of request, we can create a set of schedules fulfilling those requests, but without the timing information. We do not go into detail, but in principle, for each request, a schedule is obtained by converting each host-duration triple to its corresponding host-action, adding link-actions in between (for the corresponding links), and finally prepending an initialize-action and appending a complete-action. The resulting schedule contains no timing information, these are the variables of the model (i.e., the values sought after).

Example 14. Consider our running example. Since the user equipment generates two requests, we will need to create two schedules as follows:

$$[\mathrm{I}(r_{1_b}), \mathrm{H}_1^1(h_{0_0}, h_{0_b}, h_{0_e}), \mathrm{L}_1^{50}(l_{0_0}, l_{0_b}, l_{0_e}), \mathrm{H}_2^1(h_{1_0}, h_{1_b}, h_{1_e}), \mathrm{C}(r_{1_e})] \tag{15}$$

$$[\mathrm{I}(r_{2_b}), \mathrm{H}_1^1(h_{2_0}, h_{2_b}, h_{2_e}), \mathrm{L}_1^{50}(l_{1_0}, l_{1_b}, l_{1_e}), \mathrm{H}_2^1(h_{3_0}, h_{3_b}, h_{3_e}), \mathrm{C}(r_{2_e})] \tag{16}$$

The variables (e.g., $h_{0_0}, l_{1_b}, r_{2_e}$) represents timing information, which is to be determined.

6.1. Constraints

As seen above, given a set of requests, we can introduce a set of schedules fulfilling those requests. First, we present constrains requiring these schedules to be sound (**Sound schedules**). Next, we need to ensure that the schedules are compatible, that is, a host is not used by two schedules at the same time (**No host overlap**), and the sum of bandwidths used on a single link at any point in time does not exceed the link capacity (**No link over-utilization**). We also need to ensure that the scheduling policy is enforced, in this case-study being "first-come first-served" priority-based scheduling (**Scheduling**). Note that we also need to enforce this to routing over links (**Routing**), where priority is ignored.

We begin with some useful predicates and sets, which will be used in the remainder of this section. Furthermore, we assume that $\mathcal{A}$ is a set containing all action-time pairs.

Predicates:

$$\mathit{disjoint}(T^1, T^2) := T_E^1 \leq T_B^2 \vee T_E^2 \leq T_B^1 \tag{17}$$

$$\mathit{in}(t, T) := T_B \leq t < T_E \tag{18}$$

$$\mathit{used}(l, t, \mathrm{L}_{l'}^{bw}(T)) := \mathit{in}(t, T) \wedge l = l' \tag{19}$$

The $disjoint$ predicate is true whenever the actual executing time of the two time-tuples T^1 and T^2 does not overlap; in is true when t is in the actual executing time of T; and finally $used$ is true if link l is used by link-action $\mathrm{L}_l^{bw}(T)$ at time t.

Sets:

$$hostActions(h) := \{\mathrm{H}_{h'}^{p}(T) \mid \mathrm{H}_{h'}^{p}(T) \in \mathcal{A}, h = h'\} \tag{20}$$

$$linkActions(l) := \{\mathrm{L}_{l'}^{bw}(T) \mid \mathrm{L}_{l'}^{bw}(T) \in \mathcal{A}, l = l'\} \tag{21}$$

$$hostEnds(h) := \{t \mid \mathrm{H}_{h'}^{p}(T) \in \mathcal{A}, h = h' \wedge t = T_E\} \tag{22}$$

$$linkStarts(l) := \{t \mid \mathrm{L}_{l'}^{bw}(T) \in \mathcal{A}, l = l' \wedge t = T_B\} \tag{23}$$

$$linkEnds(l) := \{t \mid \mathrm{L}_{l'}^{bw}(T) \in \mathcal{A}, l = l' \wedge t = T_E\} \tag{24}$$

The set $hostActions(h)$ ($linkActions(l)$) contains all host-actions (link-actions) for host h (link l); $hostEnds(h)$ is the set of all times at which a host-action for host h ends executing, and finally $linkStarts(l)$ and $linkEnds(l)$ are defined analogously.

Sound schedules. First, we require that each schedule S_R begins at its start time, thus for each initialize-action $I(T)$:

$$T_0 = R_{start} \tag{25}$$

Next, no action can actually begin executing before it is ready:

$$T_0 \leq T_B \leq T_E \tag{26}$$

The duration of host-actions must be in the interval of the d_{min} and d_{max} of the corresponding action in the request, thus for all $\mathrm{H}_h^p(T)$:

$$d_{min} \leq T_E - T_B \leq d_{max} \tag{27}$$

Durations of link-actions must be equal to the latency of the link, thus for all $\mathrm{L}_l^{bw}(T)$:

$$T_E - T_B = latency(l) \tag{28}$$

We require that for all actions in a schedule no action can begin before the preceding one has ended. Hence, for each pair of successive actions $(a^1, T^1), (a^2, T^2)$ in a schedule:

$$T_0^2 = T_E^1 \tag{29}$$

Example 15. For our running example, the following constraints would be introduced:

$$r_{1_b} = 0, r_{1_b} = h_{0_0}, h_{0_e} = l_{0_0}, l_{0_e} = h_{1_0}, h_{1_e} = r_{1_e} \tag{30}$$

$$r_{2_b} = 75, r_{2_b} = h_{2_0}, h_{2_e} = l_{1_0}, l_{1_e} = h_{3_0}, h_{3_e} = r_{2_e} \tag{31}$$

$$h_{0_0} \leq h_{0_b} \leq h_{0_e}, l_{0_0} \leq l_{0_b} \leq l_{0_e}, h_{1_0} \leq h_{1_b} \leq h_{1_e} \tag{32}$$

$$h_{2_0} \leq h_{2_b} \leq h_{2_e}, l_{1_0} \leq l_{1_b} \leq l_{1_e}, h_{3_0} \leq h_{3_b} \leq h_{3_e} \tag{33}$$

$$5 \leq h_{0_e} - h_{0_b} \leq 15, l_{0_e} - l_{0_b} = 10, 10 \leq h_{1_e} - h_{1_b} \leq 20 \tag{34}$$

$$5 \leq h_{2_e} - h_{2_b} \leq 15, l_{1_e} - l_{1_b} = 10, 10 \leq h_{3_e} - h_{3_b} \leq 20 \tag{35}$$

No host overlap. We must ensure that no two host-actions of the same host overlap.

$$\forall\, \mathrm{H}^{p}_{h_1}(T^1), \mathrm{H}^{p}_{h_2}(T^2) \in \mathcal{A} \quad h_1 = h_2 \Rightarrow disjoint(T^1, T^2) \tag{36}$$

Example 16. For our running example, the following constraints (simplified by removing antecedent and replacing $disjoint$ with its definition) would be introduced (we only write those where the hosts are the same, as the others are trivially satisfied):

$$h_{0_e} \leq h_{2_b} \vee h_{2_e} \leq h_{0_b} \tag{37}$$

$$h_{1_e} \leq h_{3_b} \vee h_{3_e} \leq h_{1_b} \tag{38}$$

No link over-utilization. We must ensure that the sum of the bandwidth usage of active actions at any given point is less than the capacity of the link. However, note that if over-utilization would occur, it would at least occur when a new link-action is started:

$$\forall l \in L \quad \forall t \in linkStarts(l) \left(\sum_{\substack{\mathrm{L}^{bw}_{l}(T) \in \mathcal{A} \\ used(l,t,\mathrm{L}^{bw}_{l}(T))}} bw \right) \leq capacity(l) \tag{39}$$

Example 17. For our running example there are two points in time when a link action begins (l_{0_b}, l_{1_b}), so these two time-points must be checked. For clarity, let $load(cond, bw)$ be a function which returns bw if $cond$ is true otherwise 0. Using this, we can expand the summation sign and explicitly enumerate the two time-points:

$$load(in(l_{0_b}, T(l_{0_0}, l_{0_b}, l_{0_e})), 50) + load(in(l_{0_b}, T(l_{1_0}, l_{1_b}, l_{1_e})), 50) \leq 500 \tag{40}$$

$$load(in(l_{1_b}, T(l_{0_0}, l_{0_b}, l_{0_e})), 50) + load(in(l_{1_b}, T(l_{1_0}, l_{1_b}, l_{1_e})), 50) \leq 500 \tag{41}$$

Scheduling. As mentioned, we present a formalization of first-come first-serve scheduling of VNFs. We can ensure this by enforcing that if a host-action is not started immediately, during all time-steps between T_0 and T_B there is another computing host-action, which was ready earlier or at the same time with higher or equal priority. Note that we do not need to check all time-steps, but it is enough to check that indeed at T_0 this is the case, and then all time-steps where a host-action (of the same host) ends (since this is the only time when the host could become idle):

$$\begin{gathered} \forall(\mathrm{H}^{p^1}_{h}(T^1)) \in \mathcal{A} \quad \forall t \in hostEnds(h) \cup \{T^1_0\} \\ T^1_0 \leq t < T^1_B \Rightarrow \\ \exists \mathrm{H}^{p^2}_{h}(T^2) \in hostActions(h) \quad \mathrm{H}^{p^1}_{h}(T^1) \neq \mathrm{H}^{p^2}_{h}(T^2) \wedge in(t, T^2) \wedge T^2_0 \leq T^1_0 \wedge p^1 \geq p^2 \end{gathered} \tag{42}$$

Example 18. In our running example, there are in total four host-actions (two per request), and for each host-action there is one other host-action of the same host, thus for each host-action there are two time-points to check, the first possible time for the action, and the end-time for the other host-action. For each check, we can replace the existential

quantifier with its enumeration, which in this case is only one action. We show this for the first host-action (the constraints for the other host-actions are analogous):

$$h_{0_0} \leq h_{0_0} < h_{0_b} \Rightarrow in(h_{0_0}, (h_{2_0}, h_{2_b}, h_{2_e})) \wedge h_{2_0} \leq h_{0_0} \wedge 1 \geq 1 \tag{43}$$

$$h_{0_0} \leq h_{2_e} < h_{0_b} \Rightarrow in(h_{2_e}, (h_{2_0}, h_{2_b}, h_{2_e})) \wedge h_{2_0} \leq h_{0_0} \wedge 1 \geq 1 \tag{44}$$

Routing. Routing must also be enforced, and we use the same policy as for scheduling, "first-come first-serve". We can ensure this by enforcing that if a link-action is not started immediately, during all time-steps between T_0 and T_B there are other communicating link-actions, which were ready earlier or at the same time, consuming bandwidth such that the link-action can not be accommodated. Note that we do not need to check all time-steps, but it is enough to check that indeed at T_0 this is the case, and then all time-steps where a link-action (of the same link) ends (since this is the only time when the link could get sufficient free bandwidth):

$$\forall\, \mathrm{L}_l^{bw}(T^1) \in \mathcal{A} \quad \forall t \in \mathit{linkEnds}(l)$$

$$in(t, T^1) \Rightarrow \left(\sum_{\substack{\mathrm{L}_l^{bw'}(T^2) \in \mathcal{A}, \mathrm{L}_l^{bw}(T^1) \neq \mathrm{L}_l^{bw'}(T^2) \\ used(l,t,\mathrm{L}_l^{bw'}(T^2))}} bw' \right) > capacity(l) - bw \tag{45}$$

Example 19. The routing for our running example is similar to the scheduling constraint, except that the existence of another host-action, is replaced by summing the bandwidth usage of all active link-actions (and we use the same function *load* as in no over-utilization constraint). We show for the first link-action, the second case is symmetrical:

$$l_{0_0} \leq l_{0_0} < l_{0_b} \Rightarrow load(in(l_{0_0}, T(l_{1_0}, l_{1_b}, l_{1_e})), 50) \leq 500 \tag{46}$$

$$l_{0_0} \leq l_{1_e} < l_{0_b} \Rightarrow load(in(l_{1_e}, T(l_{1_0}, l_{1_b}, l_{1_e})), 50) \leq 500 \tag{47}$$

6.2. Queries

If all constraints above are satisfied, the underlying set of schedules are compatible. Now, we can add additional constraints corresponding to queries, that is, properties to check.

Deadline violation. To check if any single schedule violates its deadline we add:

$$\bigvee_{S_R \in \mathcal{SC}} T_E^2 - T_B^1 > R_{deadline}, \tag{48}$$

where $\mathcal{SC}$ is the set of all schedules, and $\mathrm{I}(T^1), \mathrm{C}(T^2)$ are the initialize- and complete-action of each schedule S_R, and $R_{deadline}$ is the deadline of the corresponding request.

Example 20. For our running example, the deadline query would be:

$$r_{1_e} - r_{1_b} > 75 \vee r_{2_e} - r_{2_b} > 75 \tag{49}$$

6.3. SMT solving

To practically use these semantics to determine a property for a given 5G-SO system, we create one request for each activation of the user equipment. These are then encoded as schedules, with corresponding constraints, in Z3, together with the property to be checked.

Since the deadline constraint is true if at least one schedule violates its requirement, if the resulting SMT-problem has a solution this means that there is at least one execution trace in which the system fails, thus the configuration is not a solution. Note the inverse of the result: if no SMT model is found then the 5G-SO system is safe.

7. Modeling and Verification Framework

By combining the UML profile (Sec. 4) with the semantics (Sec. 5 and Sec. 6) we obtain a framework which can be used to model various 5G-specific scenarios, and analyze them by employing a suitable back-end (e.g., UPPAAL, Z3). Our framework has two benefits: (1) a practitioner can use an industrially-accepted UML tool to model the system without knowledge of the underlying TA and first-order logic modeling, and (2) the verification results provide guarantees of the 5G-SO modeled behavior.

The intended workflow of the framework is to start from a case study and its requirements in natural language. First, the UML5G-SO is applied profile and a structural representation is created using a class diagram. Some of the classes in the diagram, referred to as active classes, have behavior modeled by UML statecharts. Next, the requirements of the case study are formalized either as (T)CTL queries or SMT formulas. This part is meant to be done by an expert working with both 5G as well as the underlying semantics.

Afterwards, a 5G expert can continue work without knowledge of formal methods. To verify a particular 5G-SO system, the class diagram is instantiated, yielding an object diagram of the system. Secondly, the semantics are applied (using TA templates or SMT formulas). Finally, the corresponding queries or formulas are analyzed to verify the system. We provide a tool, G^5, which can execute the second and third step automatically.

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, G^5[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 [28]); it can automatically generate both a TA model and an SMT model, and use a corresponding back-end to verify that all deadline requirements of slices are met. The user designs their system in an UML tool, saving an object diagram corresponding to the system they wish to analyze. Then in the G^5 tool, the diagram is loaded and by the press of a button, deadline-violations can be checked.

[5] https://github.com/ptrbman/GGGGG

7.2. Comparison of semantics

We have now presented two formalizations of the 5G system semantics, one based on TA and the other on first-order logic. They are formulated independently, but aimed at expressing the same behavior. Consider a counter-example found in the SMT-model of a particular 5G system. We can then extract all the schedules with their action-time pairs and form an execution trace of the system. In the same manner, if the deadline-query of the TA formalization is unsatisfied then there exists a counter-example that can also be seen as a trace.

We conjecture that it is always possible to translate a counter-example trace (from TA) to corresponding schedules with action-time pairs (to SMT) and vice versa. If this conjecture holds, both formalizations should always yield the same answer to the question whether a deadline is violated or not. We explore this further in the experimental evaluation.

8. Case study

As a case study, we consider 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). 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.

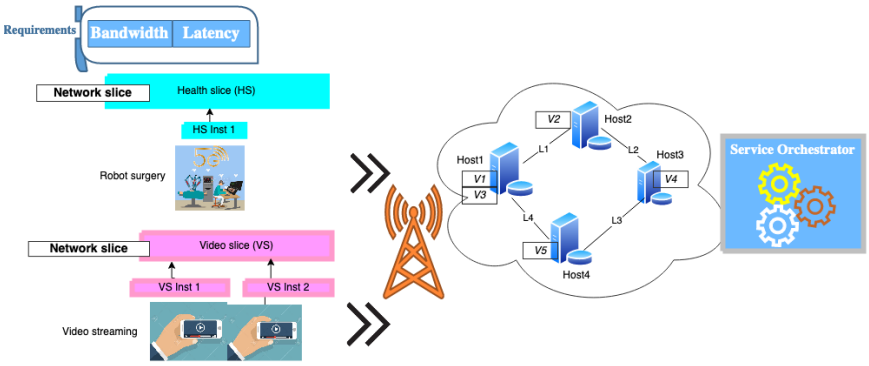

Fig. 9. Case study

We assume that we have one UE accessing the health slice and two UE accessing the video slice. The case study is depicted in Fig. 9. Slices consist of their respective VNF instances, e.g., the health slice consists of VNF sequence $[V_1, V_2]$, and the video slice has the VNF sequence $[V_1, V_3, V_4, V_5]$. Note that the slices share V_1. The overlay network comprises four hosts onto which VNFs are allocated. For instance, as shown in Fig. 9, we consider that V_1 and V_3 are allocated to Host1. Furthermore, the hosts are connected via

virtual links, e.g., Host1 and Host2 are connected via $L1$. However, not all hosts have a one-to-one link connectivity established, e.g., there is no direct link between Host1 and Host3. Hence, any routing scheme from Host1 to Host3 must use the path consisting of links $[L_1, L_2]$, or the alternative path $[L_4, L_3]$. The slices cater to different application requirements, for example while the robotic surgery application requires both low latency and high bandwidth, the video streaming application has high bandwidth requirements, but not critical latency constraints. We focus on two requirements in this case study, the end-to-end latency requirements of the slices are met and that the end-to-end bandwidth requirements of the slices are met.

8.1. Modeling with the Framework

The class diagram description of our case study is shown in Fig 10. We apply stereotypes and add attributes and functions, in the same manner as in Example 4. We model two categories of 5G user equipment, $5GCamera$ and $MobilePhone$, which access the two kinds of slices, $HospitalSlice$ and $VideoSlice$, respectively. We also apply stereotypes on $ReqControl$, $MonitorReq$ and VM in the same manner as in Example 2. We also present an object diagram modeling of the use case. In Fig. 11 all the components of the system instance is shown with their assigned attributes.

We need to define queries corresponding to the requirements. By construction, both formalizations only gives solutions respecting bandwidth requirements. For latency requirements, we use the queries in Eq. 12 for the TA model and Eq. 48 for the SMT model.

8.2. Experimental Evaluation

We conducted an experimental evaluation of our framework on the use case. All experiments are run on a Mac, with 1,4 GHz Quad-Core CPU and 16 GB RAM, using G^5.

Use Case Instantiation To create a TA model we instantiate our TA patterns described in Sec. 5, in correspondence with the object diagram (compare with Fig. 11). In order to verify the model, we assume that the network has enough request controllers and monitors to handle the requests from independent UE (i.e., we find parameters s.t. it is sufficient).

For the SMT model we instantiate the formulas described in Sec. 6. Instead of finding a sufficient number of monitors and executors, we need to create one schedule for each request (i.e., the sum of all maximum activations of all user equipment). We then generate a model as described above to check individual queries.

One crucial common factor of performance between the two models is the maximum activation times. The number of required monitors and executors is heavily dependent on these numbers (as well as the number of schedules). We set the maximum activation of all user equipment to the same value, and will refer to this as MA. To check scalability, we set this value to one and try with increasing values until timeout.

We verify the generated model against the formalized requirement, using the UPPAAL model checker version 4.1.19 [6]. 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

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

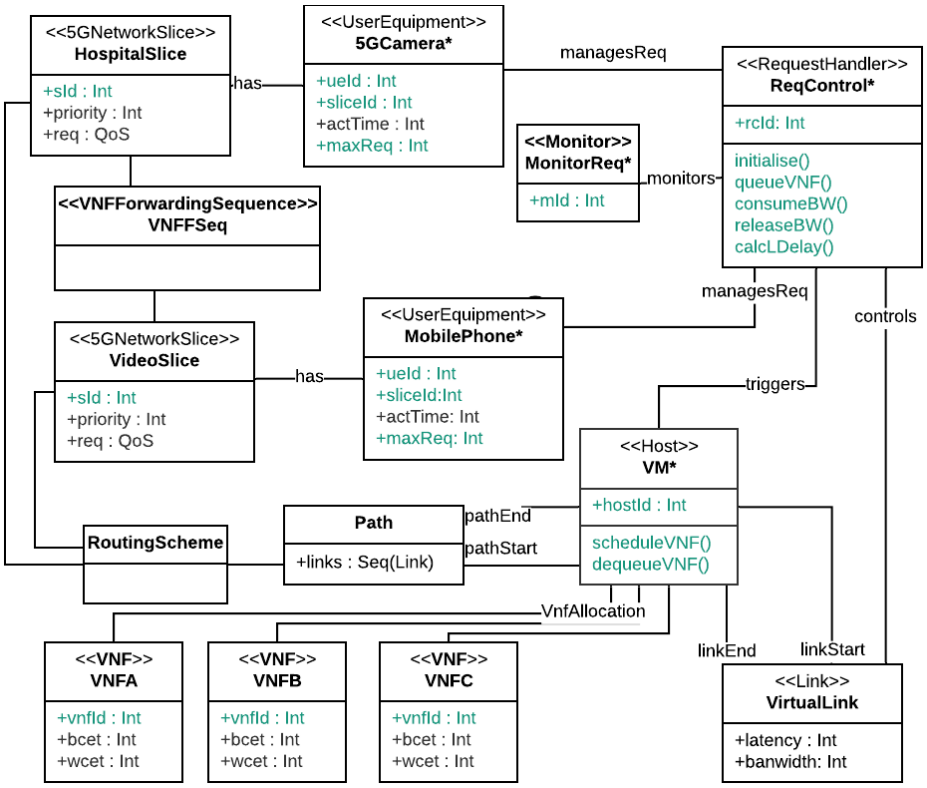

Fig. 10. Class diagram representation of our case study (It inherits all relations and multiplicities in the profile). Active classes are denoted with an asterisk.

verification fails, the mechanism allows for detecting if the model has reached any of the error locations. From Table 3, 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. As shown in Table 3 the scalability of the SMT model is better than for the TA model, but it is still non-linear (as expected, as the number of constraints are quadratic in the number of schedules).

8.3. Discussion

The use case helps us see how it is possible to draw up UML diagrams (class and object) to describe a scenario and then analyzing it using formal methods. Currently, the number of request controllers and monitors needs to be manually by the user, as we have not found a method to calculate a reasonable bound, but other than this most details of the verification process is hidden.

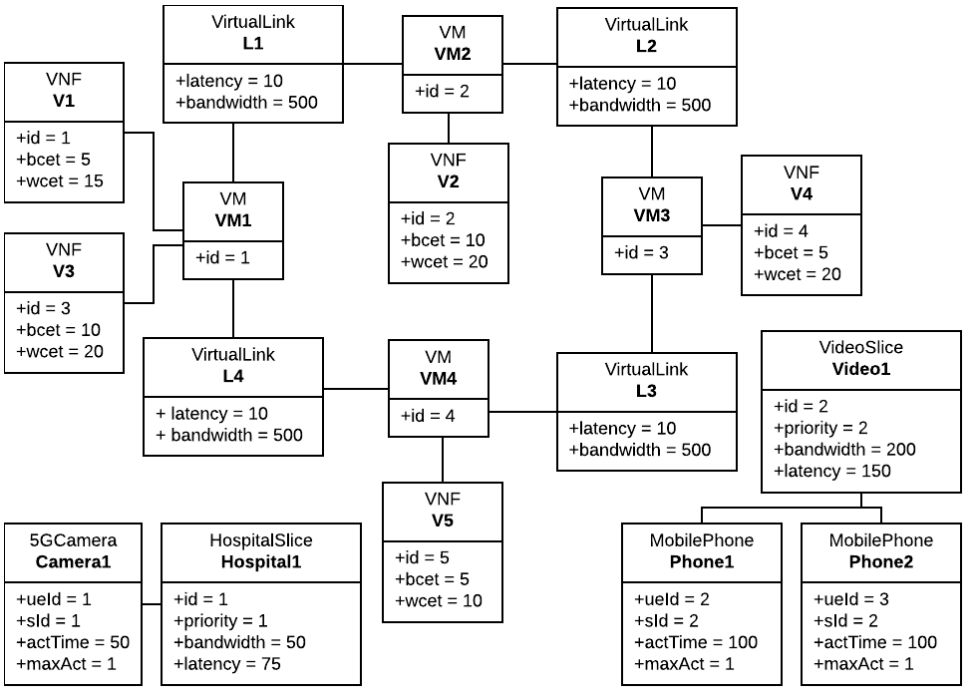

Fig. 11. Object diagram representation of the use case

The performance of both models (SMT and TA) are not enough to analyze systems with a large number of activations of user equipment. Nevertheless, it can be used to analyze systems of small size and there is room for optimizations which could greatly increase the performance. It is also important to emphasize, that in contrast to many other approaches which has a more detailed view of a similar system (see e.g., [10]), we are not aiming at just running or simulating a system but modeling it formally s.t. it is analyzable.

It is clear that the SMT model scales better than the TA model (at least for the use case). However, all these results are satisfied, and the performance could be different when compared on unsatisfiable use cases. Moreover, (T)CTL queries can be easier to work with for formulating properties, and UPPAAL allows for studying a single execution and going back and forth to analyze and make different choices.

Table 3. Results and running-times of checking deadline on the use case with both models for different maximum activations (MA). All times are in seconds.

MA	1	2	3	4	5	6	7	8	9	10	11
TA Time	31.21	T/O	-	-	-	-	-	-	-	-	-
TA Result	sat	-	-	-	-	-	-	-	-	-	-
SMT Time	0.04	0.20	0.82	2.90	6.56	70.87	46.35	903.23	1399.85	321.38	T/O
SMT Result	sat	sat	sat	sat	sat	sat	sat	sat	sat	sat	-

To help establish the equivalence of the models, we conducted a simple experiment generating 100 systems (with four hosts, five links, two slices, eight VNFs and two UE). These were tested using both models checking if their answers agree, and in every case they did. Of course this does not prove the models equivalence but shows support for it.

9. Related Work

Substantial work within the field of 5G service orchestration is aimed at providing optimal VNF placement algorithms and routing schemes [7,18,1,6]. There is also interesting work looking into scheduling of VNFs [2], and slice chain reconfiguration [16]. In [29], Yuan et. al. applies machine learning for resource allocation in network slicing, although this is at a lower level (radio) than what we are looking at in this paper. Outside the scope of 5G, there is work on allocating virtual machines among hosts in an energy-efficient manner [3], which could be adapted and integrated into a VNF placement scheme.

However, not much effort has been invested in modeling and formal analysis of 5G orchestration 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 [25] and the work by Peuster and Karl [23]. 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. [27] 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 [17], Luque-Schempp et al. investigate the use of 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 levels 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, making it less appealing to practitioners.

Even if placed outside the 5G realm, the work on modeling E-service orchestration by Petri Nets [20] could be extended to use the timed variant of the formalism together with the respective tool support (e.g. TINA model checker) to solve the 5G orchestration problem. However, our UPPAAL-supported approach benefits from the several options for diagnosis, as well as replaying counterexamples in the tool's user-friendly GUI.

In the domain of E-health, there is a proposed framework for fog-assisted monitoring of patients [10]. A prototype testing efficiency of introducing fog/nodes is created and can therefore provide empirical evidence of improvement, while we hope that with a formal methods framework like the one in this paper, such improvements can instead be proven.

The use of UML to model 5G service orchestration is not investigated much in the literature. One recent work [22] 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.

10. 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, or SMT checking, 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 and of 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 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.

Acknowledgments. This work is supported by the EU Celtic Plus/Vinnova project, Health5G - Future eHealth powered by 5G, and the KKS synergy project ACICS - Assured Cloud Platforms for Industrial Cyber-Physical Systems, which are gratefully acknowledged.

References

1. Agarwal, S., Malandrino, F., Chiasserini, C.F., De, S.: Joint vnf placement and cpu allocation in 5g. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications. pp. 1943–1951. IEEE (2018)
2. Alameddine, H.A., Qu, L., Assi, C.: Scheduling service function chains for ultra-low latency network services. In: 2017 13th International Conference on Network and Service Management (CNSM). pp. 1–9. IEEE (2017)
3. Alsbatin, L., Öz, G., Ulusoy, A.H.: Efficient virtual machine placement algorithms for consolidation in cloud data centers. Computer Science and Information Systems 17(1), 29–50 (2020)
4. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium. pp. 414–425. IEEE (1990)
5. Börger, E., Cavarra, A., Riccobene, E.: Modeling the dynamics of uml state machines. In: International Workshop on Abstract State Machines. pp. 223–241. Springer (2000)
6. Choyi, V.K., Abdel-Hamid, A., Shah, Y., Ferdi, S., Brusilovsky, A.: Network slice selection, assignment and routing within 5g networks. In: 2016 IEEE Conference on Standards for Communications and Networking (CSCN). pp. 1–7. IEEE (2016)
7. D4.1. Definition of service orchestration and federation algorithms, service monitoring algorithms. http://5g-transformer.eu/index.php/deliverables/, accessed: 2020-07-24
8. Douglass, B.P.: Real time UML: advances in the UML for real-time systems. Addison-Wesley Professional (2004)
9. Gérard, S., Selic, B.: The uml–marte standardized profile. IFAC Proceedings Volumes 41(2), 6909–6913 (2008)
10. Hu, J., Liang, W., Zeng, Z., Xie, Y., Yang, J.E.: A framework for fog-assisted healthcare monitoring. Computer Science and Information Systems 16(3), 753–772 (2019)
11. Kunnappilly, A., Backeman, P., Seceleanu, C.: Uml-based modeling and analysis of 5g service orchestration. In: 27th Asia-Pacific Software Engineering Conference, APSEC 2020, Singapore, December 1-4, 2020. pp. 129–138. IEEE (2020), https://doi.org/10.1109/APSEC51365.2020.00021

12. Kunnappilly, A., Backeman, P., Seceleanu, C.: From UML modeling to UPPAAL model checking of 5g dynamic service orchestration. In: ECBS 2021: 7th Conference on the Engineering of Computer Based Systems, Novi Sad, Serbia. pp. 11:1–11:10. ACM (2021), https://doi.org/10.1145/3459960.3459965
13. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International journal on software tools for technology transfer 1(1-2), 134–152 (1997)
14. Leivadeas, A., Kesidis, G., Ibnkahla, M., Lambadaris, I.: Vnf placement optimization at the edge and cloud. Future Internet 11(3) (2019), https://www.mdpi.com/1999-5903/11/3/69
15. Li, X., Samaka, M., Chan, H.A., Bhamare, D., Gupta, L., Guo, C., Jain, R.: Network slicing for 5g: Challenges and opportunities. IEEE Internet Computing 21(5), 20–27 (2017)
16. Liu, Y., Lu, H., Li, X., Zhao, D.: An approach for service function chain reconfiguration in network function virtualization architectures. IEEE Access 7, 147224–147237 (2019)
17. Luque-Schempp, F., Merino-Gómez, P., Panizo, L., et al.: How formal methods can contribute to 5g networks. In: From Software Engineering to Formal Methods and Tools, and Back, pp. 548–571. Springer (2019)
18. Mahboob, T., Jung, Y.R., Chung, M.Y.: Dynamic vnf placement to manage user traffic flow in software-defined wireless networks. Journal of Network and Systems Management pp. 1–21 (2020)
19. Marchetto, G., Sisto, R., Valenza, F., Yusupov, J.: A framework for verification-oriented user-friendly network function modeling. IEEE Access 7, 99349–99359 (2019)
20. Massimo Mecella, F.P.P., Pernici, B.: Modeling e-service orchestration through petri nets. In: 3rd Int. Workshop on Technologies for E-Services, TES 2002. vol. LNCS 2444, pp. 38—-47. Springer-Verlag (2002)
21. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
22. Papageorgiou, A., Fernández-Fernández, A., Siddiqui, S., Carrozzo, G.: On 5g network slice modelling: Service-, resource-, or deployment-driven? Computer Communications 149, 232–240 (2020)
23. Peuster, M., Karl, H.: Understand your chains: Towards performance profile-based network service management. In: 2016 Fifth European Workshop on Software-Defined Networks (EWSDN). pp. 7–12. IEEE (2016)
24. Ramos, M.A., Masiero, P.C., Penteado, R.A., Braga, R.T.: Extending statecharts to model system interactions. Journal of Software Engineering Research and Development 3(1), 1–25 (2015)
25. Rosa, R.V., Bertoldo, C., Rothenberg, C.E.: Take your vnf to the gym: A testing framework for automated nfv performance benchmarking. IEEE Communications Magazine 55(9), 110–117 (2017)
26. Smullyan, R.M.: First-Order Logic. New York [Etc.]Springer-Verlag (1968)
27. Spinoso, S., Virgilio, M., John, W., Manzalini, A., Marchetto, G., Sisto, R.: Formal verification of virtual network function graphs in an sp-devops context. In: European Conference on Service-Oriented and Cloud Computing. pp. 253–262. Springer (2015)
28. USE: UML-based Specification Environment. https://sourceforge.net/projects/useocl/, accessed: 2020-07-24
29. Yuan, S., Zhang, Y., Qie, W., Ma, T., Li, S.: Deep reinforcement learning for resource allocation with network slicing in cognitive radio network. Computer Science and Information Systems 18(3), 979–999 (2021)

Peter Backeman is an Associate senior Lecturer at Mälardalen University, School of Innovation, Design and Engineering at the Formal Modeling and Analysis of Embedded

systems research group. He holds a Ph.D. in Computer Science from Uppsala University, which focused on working with SMT-solver and theorem provers. Currently, he is working with how to model embedded systems and apply different kinds of formal verification techniques.

Ashalatha Kunnappilly is a Train Traction Control System Engineer at Alstom AB, Västerås, Sweden. She holds a Masters in Embedded Systems (Amrita University, 2013) and a Ph.D. in Computer Science (Mälardalen University, Sweden, 2021). Her research focuses on developing formal models and verification techniques for predictable real-time systems.

Cristina Seceleanu is Associate Professor at Mälardalen University, School of Innovation, Design and Engineering, Networked and Embedded Systems Division, Västerås, Sweden, and leader of the Formal Modeling and Analysis of Embedded Systems research group. She holds a M.Sc. in Electronics (Polytechnic University of Bucharest, Romania, 1993) and a Ph.D. in Computer Science (Turku Centre for Computer Science, Finland, 2005). Her research focuses on developing formal models and verification techniques for predictable real-time and autonomous systems. She currently is and has been involved as organizer, co-organizer, and chair of relevant conferences and workshops in computer engineering, and is a member of the Editorial Board of the International Journal on Software Tools for Technology Transfer (STTT), Springer, and the Frontiers in Computer Science: Theoretical Computer Science.

Received: October 01, 2021; Accepted: September 01, 2022.