5G Service Orchestration Supported by Model Checking - A Case Study of Health Applications

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

[1] Mälardalen University, Västerås, Sweden
first.last@mdh.se
[2] Alkit Communications AB
Mölndal, Sweden
mathias@alkit.se

Abstract. With the increased use of 5G wireless technology for communication in e-health systems, it also arises an imperative need to verify if the application requirements in terms of resource capacity, deadlines, etc. are met by the network, especially in cases of real-time critical use cases. A promising aid to address this need lies in frameworks that allow one to model and analyze such systems before they are implemented. In our previous work, we have proposed a UML profile called UML5G-Service Orchestration, backed by model checking, which allows one to verify formally the static and dynamic behavior of 5G communication using network slicing. In this paper, we extend its tool support, called G^5, which generates automatically UPPAAL timed automata models from profile-based object diagrams, to enable automatic verification of service orchestration, including fault-tolerance aspects with respect to crashed hosts or links. We chose an industrial case study of two different e-health applications that use 5G network slicing, for an evaluation of the approach, which lets us identify the factors that impact its scalability.

Keywords: UML5G-Service Orchestration · Model checking · UPPAAL.

1 Introduction

The 5th generation of cellular wireless technology, 5G, bears the promise to support a large set of e-health applications of various latency, reliability, and bandwidth requirements. To accomplish this, 5G employs a technique named *end-to-end network slicing* [15]. As the name suggests, the entire communication network is sliced into various chunks, the slices, each capable of supporting various requirements, e.g., low latency, high reliability, etc. There are four standard network slices designed by 3GPPP, namely the Ultra Reliable Low Latency Communication (uRLLC), Enhanced Mobile Broadband (eMBB), Massive Machine Type Communication (mMTC), and Vehicle to Everything, (V2X) [1], which are accessed by the corresponding 5G user equipment (UE) to meet their application requirements. In order for these network slices to support a variety

of applications, several network functions (both application- and 5G-network-specific ones) are chained in a particular manner to meet the application requirements. In this work, we consider only virtualized counterparts of network functions, that is, *virtual network functions* (VNFs).

In order to support end-to-end network slicing, the VNFs need to be allocated on virtual machines (VM). When *allocating* these VNF instances on VMs, one needs to ensure that the resource requirements of the VNFs (e.g., processing power, RAM) are met by the capacities of the respective VMs. In addition, when a UE (that is subscribed to a 5G network service provider) requests a network slice, one needs to find a *mapping* that chooses among the replicas of VNF instances available to constitute its network slice instance [6]. Next, since these VNF instances can be in different VMs, one needs a *routing scheme* across the hosts, such that the order of VNF chaining in a particular slice is respected. The routing scheme should also consider the requirements in terms of bandwidth and latency. Adding to the complexity, there also come the dynamic aspects of the network arising from simultaneous slice requests, VNF sharing, etc. In this paper, we consider the collective problem of *allocation*, *mapping* and *routing*, assuming dynamic behavior of the network as mentioned above, and refer to it as the *dynamic 5G service orchestration (5G-SO) problem* [2].

Although solutions for the 5G-SO problem do exist in the literature [22, 20], there is a lack of frameworks that enable formal verification of the solutions at design stage. In our previous work [18], we have proposed a UML-based modeling framework, namely UML5G-Service Orchestration (UML5G-SO), which facilitates modeling of the structure and behavior of 5G-SO systems, based on a new profile that we define. We also proposed a method of verifying the UML models by using the UPPAAL model checker [19], by which one can identify if a particular allocation, mapping and routing satisfy the application requirements, considering dynamic network behavior as mentioned above.

To enable model-checking of UML models for 5G orchestration, we extend and improve our frameworks' tool support, called G^5, which can automatically generate timed automata models from the corresponding UML5G-SO object diagram description of the system. The tool then uses UPPAAL as a back end, to verify if application requirements are met. The extension of the tool implements the capability of verifying k fault-tolerance of the system [10], by simulating scenarios of crashed hosts. In this paper, we also carry out an evaluation of the tool, on a case-study containing two industrial e-health applications, and gather insights with respect to factors that impact the tool's scalability.

The rest of the paper is organized as follows. In Sec. 2, we overview the preliminaries of UML 2.0 timed-automata, UPPAAL model checker, and introduce the dynamic 5G service orchestration problem. Sec. 3 details our case study and its requirements. Our G^5 tool is explained in Sec. 4, where we also provide an example of the workflow, illustrate the underlying UPPAAL models, and our experimentation results. Next, we present a brief discussion of gained insights in Sec. 5. We compare our contribution to related work in Sec. 6, and provide the concluding remarks and directions of future work in Sec. 7.

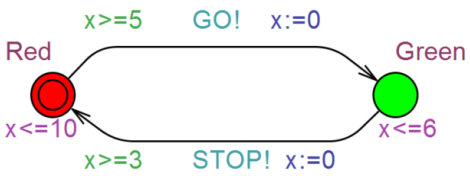


Fig. 1: An example of an UPPAAL timed automaton of a traffic light

2 Preliminaries

2.1 UML Modeling

The Unified Modeling Language (UML) [12] is a widely-used modeling language, also adopted in industry, for describing structural and behavioral artifacts of complex systems. The existing UML diagrams can be extended by *UML Profiles*, allowing the definition of *stereotypes, tagged values* and *constraints* for capturing domain-specific concepts. In our previous work [18], we have introduced a UML5G-SO profile that enabling modeling and analysis of 5G-SO configurations. For readability of our current work we include the profile description in Fig. 5. One can employ our UML5G-SO profile to create class and object diagrams depicting the structure, as well as state-chart behavior of 5G-SO systems. The framework is explained in detail in our previous work [18].

2.2 Timed Automata and UPPAAL

A *timed automaton* (TA) is a finite-state automaton extended with real-valued variables, called *clocks*, suitable for modeling real-time systems [7]. UPPAAL [8] is a tool for modeling, simulation, and model checking of real-time systems, which uses an extension of TA as the modeling formalism. Fig. 1 depicts a simple UPPAAL TA model of a traffic lights example. Two circles labeled `Red` and `Green`, called *locations*, model the two colors of the traffic light. A clock variable `x` that measures the elapse of time is used in the *invariants* (Boolean expressions over clocks) on locations (e.g., `x<=10`) to enforce an upper bound of delaying in each location, respectively. The directed lines used to connect locations are called *edges*, and can be decorated by *guards*. Guards are Boolean conditions over clocks or discrete variables, and enable traversing the respective edge once they evaluate to true. In UPPAAL, locations denoted by encircled c are called *committed* locations, and require that time does not elapse in those locations and the next edge to be traversed must start from one of them. Clocks can be reset over edges, e.g., `x:=0` in Fig. 1, whereas discrete variables can be assigned values, accordingly, via updates on the edges, or via functions that are implemented in the declaration of the TA, by a subset of the C language. A *network* of UPPAAL TA, $B_0 \parallel ... \parallel B_{n-1}$, models a parallel composition of n UPPAAL TA that can synchronize via *synchronization channels* (i.e., $a!$

is synchronized with $a?$ by handshake). In Fig. 1, the edges are labeled with channels named `STOP` and `GO`, which synchronize this TA with other TA of the model.

The UPPAAL queries that we verify in this paper are: 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.

2.3 Dynamic 5G Service Orchestration

The dynamic 5G Service Orchestration (5G-SO) has been defined in our previous work [18], and here we give a short description. First, the concept of *Overlay Network (ON)* describes a set of virtual machines connected via virtual links (directly or not), configured on top of a common physical hardware infrastructure. The *ON* is used by a set of *5G User Equipment (UE)* that requests network slices for its application. A *network slice (NS)* is defined as a sequence of *Virtual Network Functions (VNFs). VNFs* are software implementations of network functions, e.g., a firewall, which can be deployed on, e.g., virtual machines [20]. The order (sequence) in which VNFs are chained to achieve certain functionality is called *VNF Forwarding Sequence (VNFSeq).*

In order to enable network slicing and fault tolerance, we assume a service provider that provides multiple replicas of the VNF instances that are hosted in various hosts in the ON. A VNF *allocation* is an assignment of VNF instances to hosts, ensuring that the respective VNF instance resource requirements matches the resource capabilities of the host. The 5G UE subscribes to its respective service provider, and is given access to various 5G network slices it requests for. A *mapping* maps all VNFs that form a UE's NS instance to VNF instances. The final step is to define a *routing* by which a path across the hosts where the corresponding VNF instances are allocated such that the mapping is respected.

The dynamic 5G-SO problem that we consider in this paper assumes simultaneous slice requests, VNF sharing, and comprises the allocation, mapping, and routing steps together. We assume a fixed allocation, and *configurations* describe the set of possible mappings and routings. In addition, we consider that a configuration is a *solution* to our 5G-SO problem, only when it satisfies the slice requirements in terms of bandwidth and latency.

3 Case-study Description

Next, we describe in detail a case study of modeling and verifying the requirements of selected e-Health applications. Applications and system requirements are provided by industrial partners of the EU Celtic Health5G project [3].

3.1 eSense: Alkit's Home-based Rehabilitation Training System

For many medical conditions, including chronic diseases such as Chronic Obstructive Pulmonary Disease (COPD), physical exercise and rehabilitation training is of utmost importance. In order to obtain optimal effect, exercises have to be performed in a structured way and with continuity for the patient. To improve the efficiency of care and the situation for patients, home-care solutions are needed. Such a solution for home-based rehabilitation, containing planning, monitoring, feedback, follow-up and video communication services is being developed by Alkit Communications AB [26], in Sweden.

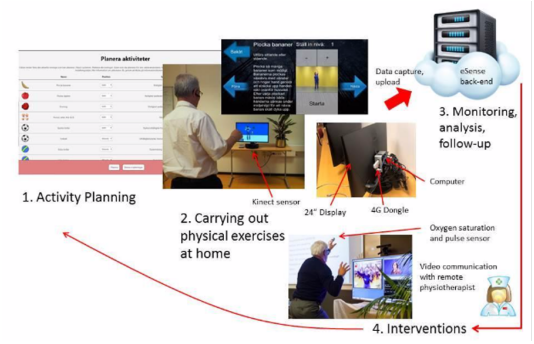


Fig. 2: eSense: Alkit's Home Rehabilation System

The system is described in Fig. 2. As shown, the home-based rehabilitation process consists of four main steps supported by the system: a) Activity planning, b) Physical exercises, c) Monitoring, analysis, and follow-up, d) Interventions. These steps are illustrated in Fig. 2. In the activity-planning phase, a medical professional (typically a physiotherapist) decides what physical exercises are suitable for a specific patient and which difficulty level is appropriate, based on the patient's health status. The plan is created in a web-based planning tool. When the planning is done, a patient can start performing the planned exercises at home, supported by an exercise home terminal based on gamification and body-tracking. The patient's performance is monitored by the system and performance metrics are calculated and uploaded to a back-end server infrastructure. The patient's status can be assessed by analyzing the uploaded data, which can trigger interventions by the medical professionals via video-mediated communication.

Application Requirements and the need for 5G Network Slicing: The main requirements of the application are: a) high bandwidth ($\geq 10Mbps$), and b) low latency ($\leq 100ms$) communication between the patients' homes and the care giver's facilities. At present, the communication infrastructure used in patients'

homes is 4G. However, the application has stringent requirements in terms of bandwidth, latency, and reliability due to using both live video and patient data, and could benefit from the emerging 5G network infrastructure and its network slicing technology that promises to guarantee such requirements. Hence, we elaborate a proposed way of employing 5G network slicing to support the application. To accomplish the latter, we consider using a 5G network slice, named *eSenseSlice* to serve our purpose. Inspired by the work of Vassilaras et. al [27] and taking into consideration the application's functionality, we define the *VNFSeq* of *eSenseSlice* as: $BBU - UPF - EPA - EC - CPF - DN(C)$, where *BBU* denotes the Base-Band Unit VNF, *UPF* is the User Plane Function, *EPA* is the Edge Processing and Caching VNF, *EC* stands for Edge Caching, *CPF* is the Control Plane Function, and $DN(C)$ is the Data Network Cloud VNF.

3.2 Wireless Hospital Patient Monitoring Using e-Health Band

Currently, continuous vital sign monitoring is being performed for patients only inside hospital intensive care units (ICUs). However, for some patients, like the patients after surgery (but not in ICUs), frequent vital sign monitoring is necessary to detect sudden changes in their health conditions. At present, these vital signs are examined by nurses in a routine basis, which is often hectic and not so effective if there is shortage of nurses. Hence, there is a need to enhance the efficiency of the vital-sign monitoring. However, in most critical scenarios, an automatic detection of the emergency scenario by continuous vital sign monitoring and automatic alerting of the caregiver is beneficial. In order to accomplish this, a reliable wireless communication coverage is also required.

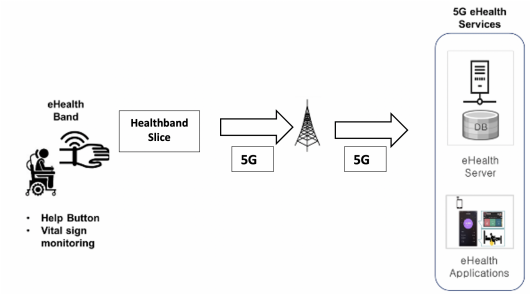


Fig. 3: e-Health Band: Wireless Patient Monitoring inside Hospitals

The wireless patient monitoring system is described in Fig. 3. The sensing devices are the e-Health bands (5G-enabled devices) that monitor the vital signs to detect emergency, and use a *Healthband* slice to push the alarm. Along with the alarm signal, they can also transmit some vital data that help the diagnosis, namely the blood type, possible allergy, etc.

VNF	CPU (GHz)	RAM (GB)	BCET(ms)	WCET (ms)
BBU	3.3	3	1	13
EPA	3	5	2	9
EC	3	6	5	8
CPF	2.2	2	3	5
UPF	2	1	2	10
DN(c)	3	8	10	15
DN(e)	3.1	5	5	10

Table 1: VNF Resource Requirements

Link	Bandwidth (MBps)	Latency (ms)	Link	Bandwidth (MBps)	Latency (ms)
L1	300	5	L5	500	25
L2	200	5	L6	300	40
L3	400	5	L7	200	10
L4	200	5	L8	400	10

Table 2: Resource Capacities of Links

Application Requirements and the need for 5G Network Slicing: The main requirements of the application are: (i) low latency (less than 150 ms for transmitting the alarm signals), and (ii) data size less than 1 MBps. In order to support such application requirements, while co-existing with many different applications that share the network, especially during critical scenarios like health parameter deviations, a *healthband* slice is employed for the communication. Just like the previous case, we define a *healthband* slice as the following VNF chain: $BBU - UPF - DN(e) - CPF$, where DN(e) is the Data Network Edge VNF, while all the other VNFs are the same as before.

The VNF resource requirements of the respective VNFs used by both applications are tabulated in Table 1. The requirements so reported are in prescribed ranges as described by the literature [21].

Host category	Processing Power (Ghz)	RAM (GB)
VNF Hosting Nodes	3.5	32
Routing Nodes	0.3	1

Table 3: Resource Capacities of Hosts

3.3 Overlay Network

To be able to analyze the case study applications mentioned above, we consider an overlay network with a set of hosts and links connecting them. We also assume that the VNF instances are allocated in multiple copies in the hosts and various slices access them on demand.

The overlay network that we employ for our analysis is depicted in Fig. 4. As shown, we have a set of eight hosts, out of which four are referred to as **Computation Hosts** (CH), CH_1 to CH_4, and four are called **Routing Hosts** (RH), that is, RH_1 to RH_4, connected via virtual links L_1 to L_8. As the name suggests, only CH have capabilities of processing VNFs, while RH are just for routing purposes. The host capabilities are tabulated in Table 3 [4], and the link capabilities are tabulated in Table 2 [6]. A 5G UE of type *eSense* or *HB*

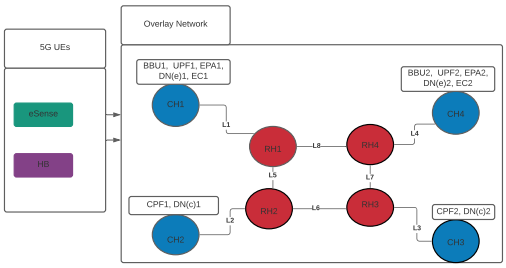


Fig. 4: Case-study Scenario

accesses this overlay network while requesting its respective slice. In this paper, we assume that each UE can be activated multiple times, but there exists a limit on the maximum number of activations. We also assume that the same category of UE and their activations share the respective slice instances. For example, all *eSense* UE and their activation share an *eSense* network slice instance, say, $BBU_1 - UPF_1 - EPA_1 - EC_1 - CPF_1 - DN(c)_1$, and all healthbands and their activation share the same healthband slice instance with the *VNFSeq* given as $BBU_2 - UPF_2 - DN(e)_1 - CPF_2$.

As mentioned previously, we begin by assuming an allocation of the VNF instances to various hosts as shown in Fig. 4. The allocation respects the resource constraints and capabilities of both VNFs and hosts. In our earlier work [17], we describe how one can verify the resource allocation using object constrained language (OCL) queries in the USE tool. Given the allocation, we now exemplify a routing scenario of Slice 2, which we later verify to see if it can meet the application requirements. Since BBU_2 and UPF_2 are located in the same host, that is, CH_4, there is no routing in the first step, however since the next VNF instance, $DN(e)_1$, is located in CH_1, we need to find a routing path between CH_4 and CH_1, and since there are no direct link connections, we need to traverse RH_1 and RH_4 through links L_1, L_8 and L_4. After executing $DN(e)_1$ in CH_1, we need to find the route to reach CH_3, where CPF_2 is located in CH_4. There are two options for this: a) a path through RH_1, RH_4, RH_3, using links L_1, L_8, L_7, L_3, or b) a path through RH_1, RH_2, and RH_3, using links L_1, L_5, L_6, L_3. Whichever route we choose, we need to ensure that the respective links have enough bandwidth capacities to serve the request from Slice 2, as well as ensure that the overall

VNF processing time in hosts and their respective routing latency do not affect the end-to-end latency requirements. This analysis is often time consuming, and most often impossible to carry out manually. Hence, in this paper, we propose the G^5tool that can automate such analysis, while ensuring solution correctness via back-end model checking.

4 Modeling and Verification Using the G^5 Tool

In this section, we show how our framework and prototype tool can be used to gain formal-verification-based insight into the case-study presented above. In previous work [18], we have introduced a first version of the G^5 tool[3] to demonstrate how one can design software, allowing engineers to interact with verification tools without any knowledge of formal methods. In this paper, we present an updated and extended version of the tool, with features including:

- Finding a mapping: with the click of a button the user can ask the G^5 tool to search for a mapping that fulfills all requirements of the provided instance. This is effectively done by enumerating all possible mappings (and routings), and stopping when a solution is found.
- Verifying properties: certain provided queries can be checked on a system (with a mapping), e.g., checking that the system does not deadlock, as well as other invariance and liveness properties. These queries can be verified on TA models that are checked by the UPPAAL model checker.
- K-fault tolerance checking: the tool supports checking fault-tolerance by removing one host (at a time) and seeing if the resulting system still has a feasible configuration. The method used by G^5 is to generate each subsystem (with a host removed) one-by-one, and do the same kind of search as in finding a mapping.

An overview of the steps involved in using the tool is shown in Fig. 6 (a).

4.1 Example Workflow

To demonstrate how the tool can be used, we consider the case study from Section 3. We begin by creating the overlay network (with hosts, links, VNFs, user equipment and slices) in USE, a UML-based specification environment [5]. An excerpt (showing hosts and links) is shown in Fig. 8. The next step is to load the instance (as a soil-file) in the G^5 tool. Our object diagrams have an associated behavioral representation that is captured by a restricted version of UML state chart, referred to as restricted state charts (RSC) (which are predefined as patterns and have TA semantics, see Sec.4.2). This enables one to automatically generate the corresponding TA models from the UML counterparts. Once the TA models are generated, one can *find* a mapping such that the system is

[3] `https://github.com/ptrbman/GGGGG`

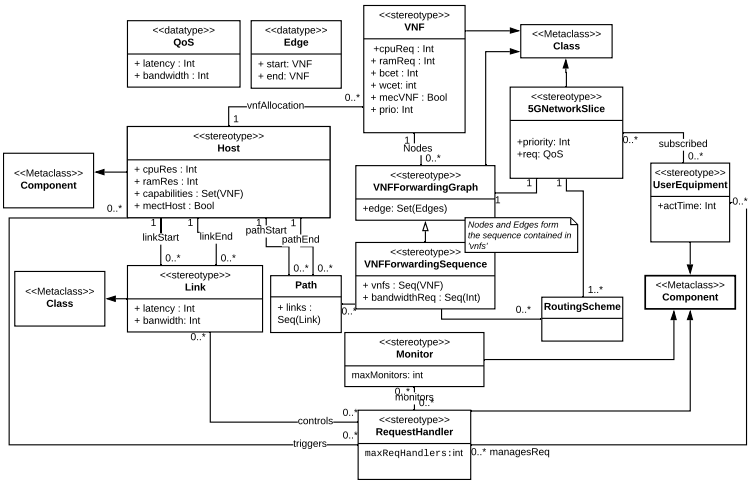
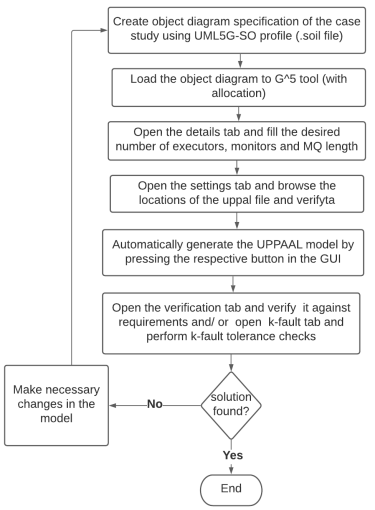
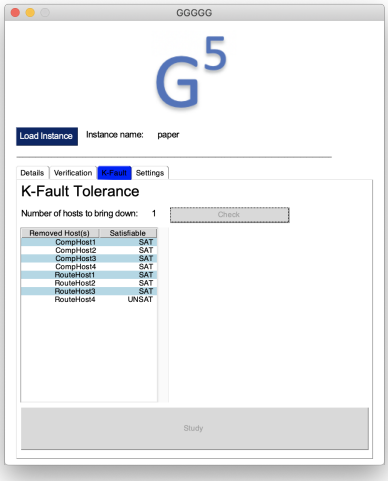


Fig. 5: UML5G-SO Profile [18]



(a) Flow chart depicting the usage of the G^5 tool



(b) Screenshot of the G^5 tool after running k-fault tolerance check.

Fig. 6

feasible, that is, in which all deadlines are met. We use G^5 for this and find the following mapping:

$$S_1 : BBU_1 \rightarrow UPF_1 \rightarrow EPA_1 \rightarrow EC_1 \rightarrow CPF_2 \rightarrow DN(c)_2$$

$$S_2 : BBU_1 \rightarrow UPF_1 \rightarrow DN(e)_1 \rightarrow CPF_1$$

We can *verify* that the given mapping fulfills the pre-defined queries, e.g., check if each request is eventually serviced (leads to), all requests are completed within their respective latency and bandwidth bounds (invariance), no deadlock etc., and see that this indeed is the case. More interestingly, we can check the *k-fault* tolerance [10] of a system. This will show us that if $RouteHost_4$, assumed crashed, is removed, the resulting system has no valid configuration (see Fig. 6 (b)). It is interesting that any one computation node can fail, and the system can still work, but if a particular routing host is removed the system fails. This indicates that an extra link could allow the system to gain a k-fault tolerance with k = 1. Indeed, if we add a link from $RouteHost_1$ to $RouteHost_3$ and re-run the k-fault tolerance check, all cases are denoted as SAT.

4.2 TA Semantics

In order to check requirements on a specific system, we need *semantics* defining how the system should behave. As described in previous work [18] this is performed by assigning a TA template to each UML restricted state chart (RSC) pattern that models the intended behavior. The RSC patterns that we use to capture behavior are relatively close to TA, despite lacking explicit clocks to measure time. Table 5 shows how some state-chart and TA concepts relate to each other and in Fig. 7 we show a RSC model and its corresponding TA counterpart. While the translation is mostly straightforward there are some differences. For instance, while the states of the state charts do form the locations in TA, pseudo-states are excluded while forming the TA. Moreover, additional locations are included that aid synchronization and capturing errors.

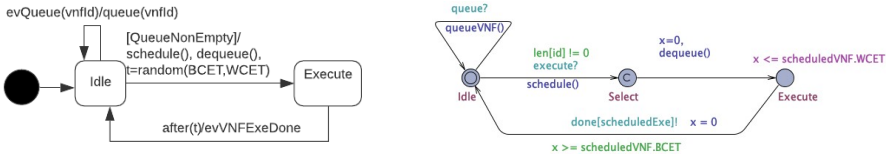
Given the TA semantics, it is then possible to encode queries in TCTL (Timed Computation Tree Logic) which can be verified in UPPAAL. We give some examples in Table 4.

Natural Language	TCTL Query
A generated request is eventually served.	$RC1.rq \rightsquigarrow RC1.rqComplete$
No request has a missed deadline.	$A\Box\ not\ MO1.MissedDeadline$

Table 4: TCTL example queries

An important artifact of this translation is that two extra arguments must be provided to the process as a whole: the number of executors and maximum queue length. The former refers to a component that handles each UE request, and a bound must be provided, which is not lower than the maximum number of

parallel requests of the system. The latter refers to the queue length of each host VNF queues, respectively, and must also not be lower than the maximum number of queued tasks for all hosts. As in bounded model checking, a too low value will yield incompleteness, but a too high value becomes a scalability problem.



(a) UML state chart model (b) TA model

Fig. 7: Modeling of the Computation Host

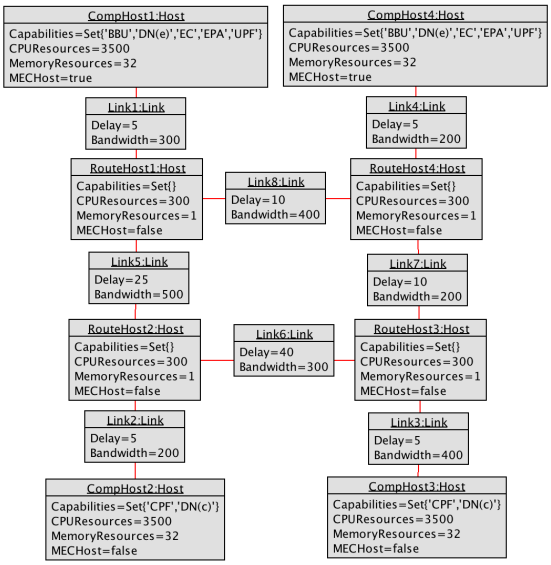


Fig. 8: Overlay Network as shown in the USE tool.

4.3 Scaling

When checking our system, we have identified some factors that potentially have an impact on the verification's run-time: (a) number of executors (b) length of message queues (c) number of activations of user equipment (d) number of user equipment. These factors have to be provided by the user and are artifacts

UML State Chart	**UPPAAL TA**
States	*Locations*
Edges	*Edges*
Parameters	*Variables*
Call events	*Synchronization channels*
Time events	*Invariant+ Guard*

Table 5: Relationship between TA and UML state chart notions

of the modeling, with no clear counterpart in the original system. Therefore, "good" values must be found for these parameters — good in the sense that they should prove sufficient to ensure that all faulty scenarios are found, but not too large to hinder the verification. We present here an experimental evaluation, to assess the impact of these factors. We do this by searching for an allocation in a modification of the example presented above (obtained by removing the second user equipment) in different scenarios by: (a) only increasing the executor count, (b) only increasing length of message queues, (c) increasing the number of activations and executors, and (d) increasing the number of user equipment (performed by just copying one of the existing ones) and executors. In (c) and (d) the number of executors must be increased as otherwise the system would not work as intended.

Note that in scenario (a) and (b), the set of solutions is constant, while in (c) and (d) the solutions are reduced (as the load of the system increases). As upper bound, 100 and 1000 were chosen for the number of executors and queue length, respectively. For the activations and user equipment, the upper bound was increased until memory-out was reached (thus halting the search). The results for (a) and (b) are presented in Fig. 9, and for (c) and (d) in Table 6.

	1	2	3	4	5	6
Activations	0.09	0.11	1.3	66.7	2906.2	M/O
User Equipment	0.4	34.2	679.0	M/O	-	-

Table 6: Runtime (in seconds) for Nr. of activations and user equipment.

We can see that the increase in run-time with an increased queue length is minor, but that the number of executors greatly affects the complexity. It is also clear that the underlying model checking does not scale well with an increasing number of user activations and user equipments, as already a value of six activations or four user equipments leads to a memory out.

5 Discussion

Our major contribution is a framework backed by tool support that allows a 5G engineer to model the system in the UML world, without needing to know anything about the underlying formal model, thereby improving the usability of

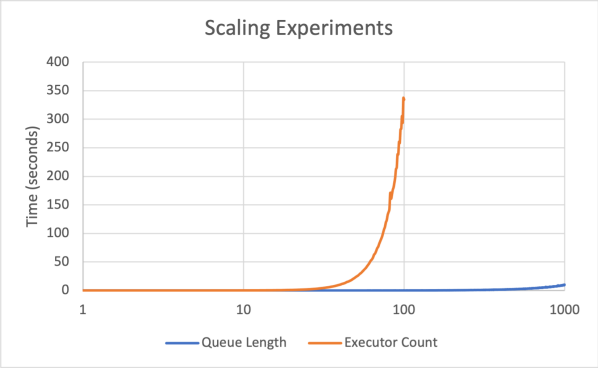


Fig. 9: Scaling of Queue length and Executor count.

the approach. Our approach allows to verify by model checking if a particular configuration of a 5G-SO system can meet application requirements in terms of bandwidth and latency. Another salient feature of our tool is that we are able to also verify the fault-tolerant behavior of the system, by implementing k-fault detection. These features are novel and unique, since no other automated approach to date provides them.Note that when considering this approach, we are only interested in finding a remapping option, without altering the initial VNF allocation that we have assumed. There are also approaches that would consider a reallocation of VNFs, however compared to a remapping, the former might be a costly option.

The analysis technique that we employ is exhaustive model-checking that provides guarantees over all possible system behaviors. However, exhaustive model-checking does not offer the level of scalability demanded by 5G networks, as the technique suffers from the well-known state-space explosion problem. Due to this, we are limited by the number of parallel UE requests that we can analyze in our system. Currently, our G^5 tool can handle one UE with a maximum of three activations, two different UEs with one activation each, or one UE with two activations and the 2nd UE with one activation. We acknowledge that this is a small number for a complex 5G network setting. Nevertheless, this paper introduces the only current tool backed by model checking that is applied on an industrial eHealth use case that uses 5G network slicing.

To mitigate the scalability of model checking, there is a multitude of ways, so we mention some of them in the following. One solution can come by optimizing our TA models. We find that a major source of our model's complexity stems from the use of executor TA that handle parallel requests. For each request, we need a free executor to handle it, yet adding them in large numbers in our TA model makes the latter too large to be handled. Hence, we could investigate how can one eliminate such executor TA in the model, and incorporate other ways to handle parallel requests. Moreover, currently our UE TA can pick up any free executor available, providing multiple parallel options to choose from,

which also contributes to exploding the state space. In future tool refinements, we can avoid this by providing a rule to pick a particular executor, say each UE picks up the executor with the lowest id. In addition, if we are interested in worst-case analysis, we can ignore the BCET of the VNFs, which will optimize the TA models and reduce the state space.

Another source of complexity that we can remove in future work is the fact that we are using enumeration of all multiple routing options, checking them one by one, which makes the verification's complexity explode, based on the many TA models that we need to verify, as each routing configuration has a corresponding TA model that we need to check. Therefore, one possible way around is using a more intelligent solution, say by using artificial intelligence to generate only the best possible mappings, for instance, using the approach in [16]. This reduces the number of configurations that we need to analyze, thereby improving scalability. Moreover, if scalability is more important than providing exhaustive guarantees, one can employ techniques like statistical model checking, e.g. UPPAAL SMC [9] which can provide probabilistic guarantees to the system. Although not exhaustive, such methods can handle complex systems.

6 Related Work

In the literature, there exists a lot of work aimed at providing optimal solutions to the 5G-SO systems. However, to the best of our understanding, not much efforts have been invested in building design-time analysis frameworks that allow a 5G engineer to model such a system and analyze if a particular solution can meet the application requirements, taking into account a variety of dynamic inputs like varying network load, VNF sharing, rerouting options, etc. (at the level of abstraction that we are interested in).

However, in recent years, there have been some endeavors focused on providing modeling and analysis support for network slicing, VNF chaining, etc. One such framework is the Gym framework [25] and work by Peuster and Karl [24], which model the description of VNFs and VNF chaining in isolation, and analyze if application requirements are met. In another interesting work [14], the authors propose a mathematical model for network slicing based on combinatorial designs such as Latin squares and rectangles and their conjugate forms. Although the approach is supported by mathematical modeling to support network slicing, due importance is given only to network slice description and providing optimized solutions to maximize the utilization of the network components, and decrease the average delay from slice request to slice activation. In contrast to our work, these approaches model VNFs, their chaining, and network slicing at a different abstraction level, without considering a system perspective, or 5G-specific scenarios, and have a different focus compared to ours. Another interesting work in the literature is the NESMO framework (Network Slicing Management and Orchestration) [11]. The framework automates network slice design, deployment, configuration, activation, and lifecycle management in multiple network infrastructure resource domains. Although this work considers a lot of intrinsic factors

that we do not consider in our framework, the aim is not to verify any 5G-SO solutions, so it does not solve the same problem as ours.

Using UML to model and analyze 5G-SO solutions is, comparatively, a less explored field. In a recent work [23], the authors model 5G network slices, namely, resource-driven, service-driven, deployment-driven views, using different UML diagrams. However, the modeling is not backed by formal verification like we present in this paper. Recently, there has also been work that uses various UML diagrams to depict the information model definitions for representing the manageable characteristics of the managed entities (so-called Network Resource Models [NRMs]) [13]. However, unlike our work, the focus of this work is on automating RAN slicing, and not on providing a UML-based framework backed by model checking, which allows a 5G engineer to model and verify 5G-SO solutions.

7 Conclusions and Future Work

In this work, we present improved tool support for modeling and verifying 5G-SO configurations, which we evaluate on a relevant industrial case-study of e-Health applications. The tool relies on UML modeling that is empowered by UPPAAL model checking. Our tool, G^5, enables a 5G engineer who does not necessarily have prior knowledge of formal models and model checking to verify the UML models of 5G orchestration, by automatically generating underlying formal TA models that are checked by UPPAAL. Applying the tool on the case study has lead to gaining useful insight on the approach, as well as its limiting factors.

Since we are using exhaustive model-checking (which is prone to state space explosion problem) to analyze our models, currently our framework may not scale to larger case studies. However, we have provided ideas to tame scalability of verification, either by optimizing our UPPAAL models to reduce the state space, or by employing artificial intelligence techniques, like reinforcement learning or evolutionary algorithms, to guide the search and generate only an optimal subset of the viable SO configurations that need to be analyzed under current scenarios. This could reduce the system workload and speed up model checking, as well as improve its scalability. Another way is to employ statistical model-checking instead of exhaustive model-checking, which can handle large systems, at the expense of verification exhaustiveness and exact guarantees. Since end-to-end network slicing in 5G systems is not yet fully standardized and available, some improvements of our framework and tool might be needed in the future. However, since our UML5G-SO profile can be easily extended, we do not foresee significant issues with extending the framework.

Acknowledgement

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

References

1. 5G evolution: 3GPP releases 16 & 17 overview. https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/5g-nr-evolution, accessed: 2020-07-24
2. 5G Transformer Project. http://5g-transformer.eu, accessed: 2020-07-24
3. EU Celtic Plus Health5G Project. http://health5g.eu, Accessed: 2021-05-14
4. Open Air system Requirements. https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/OpenAirSystemRequirements, Accessed: 2021-05-17
5. USE: UML-based Specification Environment. https://sourceforge.net/projects/useocl/, accessed: 2020-07-24
6. 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)
7. Alur, R., Dill, D.: Automata for Modeling Real-time Systems. In: Automata, languages and programming, pp. 322–335. Springer (1990)
8. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. vol. 3098, pp. 87–124. Springer (2004)
9. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal smc tutorial. International Journal on Software Tools for Technology Transfer **17**(4), 397–415 (2015)
10. Deng, W., Karaliopoulos, M., Mühlbauer, W., Zhu, P., Lu, X., Plattner, B.: k-Fault tolerance of the Internet AS graph. Computer Networks **55**(10), 2492–2503 (2011)
11. Devlic, A., Hamidian, A., Liang, D., Eriksson, M., Consoli, A., Lundstedt, J.: Nesmo: Network slicing management and orchestration framework. In: 2017 IEEE International Conference on Communications Workshops (ICC Workshops). pp. 1202–1208. IEEE (2017)
12. Douglass, B.P.: Real time UML: advances in the UML for real-time systems. Addison-Wesley Professional (2004)
13. Ferrús, R., Sallent, O., Pérez-Romero, J., Agustí, R.: On the automation of ran slicing provisioning: solution framework and applicability examples. EURASIP Journal on Wireless Communications and Networking **2019**(1), 1–12 (2019)
14. Gligoroski, D., Kralevska, K.: Expanded combinatorial designs as tool to model network slicing in 5g. IEEE Access **7**, 54879–54887 (2019)
15. Group, F.T.A.C.G.I.W.: 5g network slicing white paper. https://transition.fcc.gov/bureaus/oet/tac/tacdocs/reports/2018/5G-Network-Slicing-Whitepaper-Finalv80.pdf, accessed: 2020-07-24
16. Gu, R., Enoiu, E., Seceleanu, C., Lundqvist, K.: Verifiable and scalable mission-plan synthesis for autonomous agents. In: International Conference on Formal Methods for Industrial Critical Systems. pp. 73–92. Springer (2020)
17. Kunnappilly, A., Backeman, P., Seceleanu, C.: UML-based Modeling and Analysis of 5G Service Orchestration. The 27th Asia-Pacific Software Engineering Conferemce (APSEC) (2020)
18. Kunnappilly, A., Backeman, P., Seceleanu, C.: From UML Modeling to UPPAAL Model checking of 5G Dynamic Service Orchestration. In: 7th international Conference on the Engineering of Computer Based Systems. ACM (May 2021)
19. 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)
20. Leivadeas, A., Kesidis, G., Ibnkahla, M., Lambadaris, I.: Vnf placement optimization at the edge and cloud. Future Internet **11**(3) (2019). https://doi.org/10.3390/fi11030069, https://www.mdpi.com/1999-5903/11/3/69

21. 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)
22. Marchetto, G., Sisto, R., Valenza, F., Yusupov, J.: A framework for verification-oriented user-friendly network function modeling. IEEE Access **7**, 99349–99359 (2019)
23. 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)
24. 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)
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. Rydmark, Broeren, Jalminger, Johansson, Johanson, Ridderstolpe: Remote communication, examination and training in stroke, Parkinson's and COPD care. In: Proc. 11th Intl Conf. Disability, Virtual Reality & Associated Technologies. vol. 1, pp. 1017–1022. IDCVRAT (2016)
27. Vassilaras, S., Gkatzikis, L., Liakopoulos, N., Stiakogiannakis, I.N., Qi, M., Shi, L., Liu, L., Debbah, M., Paschos, G.S.: The algorithmic aspects of network slicing. IEEE Communications Magazine **55**(8), 112–119 (2017)