Checking Modal Contracts for Virtually Timed Ambients

Einar Broch Johnsen, Martin Steffen, Johanna Beate Stumpf, and Lars Tveito

University of Oslo, Oslo, Norway
{einarj,msteffen,johanbst,larstvei}@ifi.uio.no

Abstract. The calculus of virtually timed ambients models timing aspects of resource management for virtual machines. With nested virtualization, virtual machines compete with other processes for the resources of their host environment. Resource provisioning in virtually timed ambients can be formalized by extending the capabilities of mobile ambients to model the dynamic creation, migration, and destruction of virtual machines. This paper introduces a logic to define modal contracts regarding resource management for virtually timed ambients. Service-level agreements are contracts between a service provider and a client, specifying properties that the service should fulfill with respect to quality of service (QoS). The proposed modal logic supports QoS statements about the resource consumption and nesting structure of a system during the timed reduction of its processes. Besides a formal definition of the logic, the paper provides a corresponding model checking algorithm and its prototype implementation in rewriting logic.

1 Introduction

In cloud-computing, a service-level agreement is an official commitment or contract between a cloud-service provider and a client. Service-level agreements are offered by service providers to specify the services that should be provided to the customer as well as properties the system has to satisfy with respect to quality of service, such as mean time between failures, responsibility for various data rates, resource consumption, etc. *Quality of service* (QoS) approaches in cloud computing have recently been surveyed [1], confirming that open challenges remain which require further research to provide trustworthy cloud computing services that deliver appropriate QoS. This paper provides a formalization to support QoS statements via modal contracts for virtually timed ambients.

The calculus of *virtually timed ambients* [22] is a calculus of explicit resource provisioning, based on mobile ambients [10]. It can be used to model nested virtualization in cloud systems. Virtualization technology enables the resources of an execution environment to be represented as a software layer, a so-called virtual machine. *Nested virtualization* [19] is a crucial technology to support the heterogeneous cloud [17], as it enables virtual machines to migrate between different cloud providers [38]. It is also necessary to host virtual machines with operating systems which themselves support virtualization [7], such as Microsoft

Windows 7 and Linux KVM. The time model used to realize resource provisioning for virtually timed ambients is called *virtual time*. Virtual time is provided to a virtually timed ambient by its parental ambient, similar to the time slices that an operating system provisions to its processes. When considering multiple levels of nested virtualization, virtual time becomes a *local* notion of time which depends on a virtually timed ambient's position in the nesting structure. Virtually timed ambients are mobile, reflecting that virtual machines may migrate between host virtual machines. Observe that such migration affects the execution speed of processes in the migrating virtually timed ambient, as well as in the virtually timed ambient which is left, and in the virtually timed ambient which is entered.

This paper defines *modal contracts* which capture QoS statements for cloud systems modeled in virtually timed ambients. As virtually timed ambients can model nested virtualization in cloud systems, the modal contracts provide information on the resource consumption and nesting structure of a system of virtually timed ambients during the timed reduction of its processes. Modal contracts are formalized as properties in modal logic that a system has to satisfy. The modal logic we consider combines modal logic for mobile ambients with notions based on metric temporal logic to obtain a modal logic for virtually timed ambients. Modal logic for mobile ambients [9] enables us to make statements about the behavior of ambient systems during their reduction. Timing constraints on modalities are introduced in metric temporal logic [24,31,32], which is an extension of linear temporal logic.

To prove that a system satisfies a given modal contract, we define a simple *model checking algorithm*. We further contribute a *prototype-implementation* of the model checker in Maude [16], which is a formal specification and programming system based on rewriting logic [27].

Contributions. The main contributions of this paper are the following:

- we combine *modal logic* for mobile ambients with notions based on *metric temporal logic* in order to capture the special features of virtual time and resource provisioning in virtually timed ambients;
- we show that the resulting logic is a *conservative extension* of the modal logic for the ambient calculus, preserving satisfiability;
- we define a *model checking algorithm* for this modal logic, and develop a prototype implementation in the rewriting logic system *Maude*; and
- we illustrate all concepts by *examples*.

To the best of our knowledge, this is the first implementation of modal logic for mobile ambients in rewriting logic, and the first implementation of a model checker for mobile ambients considering time or resources.

Paper overview. We introduce virtually timed ambients in Section 2. Section 3 considers modal logic for such ambients. Section 4 introduces a model checker algorithm. Section 5 presents the implementation of the model checker in rewriting logic. We discuss related work and conclude in Sections 6 and 7.

	n	name
	tick	virtual time slice
Timed processes:		
$P, Q ::=$	$\mathbf{0}$	inactive process
$\mid$	$P \mid Q$	parallel composition
$\mid$	$(\nu n)\, P$	restriction
$\mid$	$!C.P$	replication
$\mid$	$C.P$	prefixing
$\mid$	$n[\textsc{Sched} \mid \texttt{tick}^x \mid P]$	virtually timed ambient
Timed capabilities:		
$C ::=$	$\mathbf{in}\; n$	enter n and adjust the local scheduler there
$\mid$	$\mathbf{out}\; n$	exit n and adjust the local scheduler on the outside
$\mid$	$\mathbf{open}\; n$	open n and adjust own local scheduler
$\mid$	$\mathbf{c}$	consume a resource

Table 1. Syntax of virtually timed ambients, $x \in \mathbb{N}_0$.

2 Virtually Timed Ambients

Virtually timed ambients [22,23] is a calculus of explicit resource provisioning, based on mobile ambients. Mobile ambients [10] are processes with a concept of location, arranged in a dynamically evolving hierarchy. Interpreting these locations as places of deployment, virtually timed ambients extend mobile ambients with notions of virtual time and resource consumption. The timed behavior depends on the one hand on the *local* timed behavior, and on the other hand on the placement or deployment of the virtually timed ambient or process in the hierarchical ambient structure. Virtually timed ambients combine timed processes and timed capabilities with the features of mobile ambients.

Definition 1 (Virtually timed ambients). *The syntax of virtually timed ambients is given by the grammar in Table 1.*

Timed processes differ from mobile ambients in that each virtually timed ambient contains, besides possibly further (virtually timed) subambients, a *local scheduler*. In the sequel, we omit the qualification "timed" or "virtually timed", when speaking about processes, capabilities, or ambients when the context of virtually timed ambients is clear. In the calculus, the locations for processes, called *virtually timed ambients*, are represented by names, and time slices are written as `tick`. The inactive process $\mathbf{0}$ does nothing. The parallel composition $P \mid Q$ allows both processes P and Q to proceed concurrently, where the binary operator $\mid$ is commutative and associative. The restriction operator $(\nu n)P$ creates a new and unique name with process P as its scope. Replication of processes is given as $!C.P$. A process P located in an virtually timed ambient named n is written $n[\textsc{Sched} \mid \texttt{tick}^x \mid P]$, where $\texttt{tick}^0 \equiv \mathbf{0}$. Ambients can be nested, and the nesting structure can change dynamically, this is specified by prefixing a process with a *capability* $C.P$. *Timed capabilities* extend the capabilities of mobile ambients by including a *resource consumption* capability **c** and by giving the *opening*, *exiting*, and *entering* capabilities of mobile ambients a *timed interpretation*. These capabilities restructure the hierarchy of an ambient system, so the behavior of local schedulers and resource consumption changes, as these depend on the placement of the timed ambient in the hierarchy.

The semantics of virtually timed ambients is given as a reduction system. The *reduction* relation $P \twoheadrightarrow Q$ for virtually timed ambients is captured by the rules in Tables 2 and 3. The rules for structural congruence $P \equiv Q$ are equivalent to those for mobile ambients (and thus omitted here). The rules in Table 2 make use of *observables,* also known as *barbs.* Barbs, originally introduced for the π-calculus [28], capture a notion of immediate observability. In the ambient calculus, these observations concern the presence of a top-level ambient whose name is not restricted. Let $\widetilde{m}$ describe a tuple of names, then the observability predicate $\downarrow_n$ or "barb" is defined as follows:

Definition 2 (Barbs, from [25]). *A process P strongly barbs on a name n, written $P\downarrow_n$, if $P \equiv (\nu\widetilde{m})(n[P_1] \mid P_2)$, where $n \notin \{\widetilde{m}\}$.*

A process that does not contain ν-binders is considered to be *ν-binder free.* By moving the ν-binders to the outside and only considering the inside of their scope, we can observe the bound ambients inside the scope of the ν-binders.

Definition 3 (Timed top-level ambients). *For a process P, let $P_\downarrow$ denote the set of all* timed top-level ambients*: $P_\downarrow = \{n \mid P \equiv (\nu\widetilde{m})P' \wedge P'$ is ν-binder free $\wedge\, P'\downarrow_n \wedge\, speed_n > 0\}$.*

In a virtually timed ambient, the local scheduler is responsible for triggering timed behavior and local resource consumption. Each time slice emitted by a local scheduler triggers the scheduler of a subambient or is consumed by a process as a resource in a preemptive, yet *fair* way, which makes system behavior sensitive to co-located virtually timed ambients and resource consuming processes.

Definition 4 (Local and root schedulers). *Let the variables unserved and served denote sets containing names of virtually timed ambients as well as processes (represented directly, lacking names). A* local scheduler *is denoted by*

$$\text{SCHED}_{speed}\{in, out, rest, unserved, served\},$$

where $speed \in \mathbb{Q}$ relates externally received to internally emitted time slices; $in \in \mathbb{N}$ records the number of received time slices; $out \in \mathbb{N}$ records the number of time slices to be distributed for each incoming time slice, while $rest \in \mathbb{N}$ records additional distributable time slices depending on the speed; and unserved contains local ambients with a positive speed and processes which are intended to receive one time slice in this round of the scheduling, while served contains processes scheduled for the next round.

Root schedulers*, represented as $\text{SCHED}^\dagger\{in, out, 0, unserved, served\}$, are local schedulers which do not need an input to distribute time slices and therefore have no defined speed.*

$$\frac{\begin{array}{l}\textsc{Sdl}_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k, S_k\},\quad n \in U_k \cup S_k\\ \textsc{Sdl}_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m, S_m\}\\ \textsc{Sdl}_n = \textsc{Sched}_{speed_n}\{in_n, out_n, rest_n, U_n, S_n\}\\ \textsc{Sdl}'_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k \setminus \{n\}, S_k \setminus \{n\}\}\\ \textsc{Sdl}'_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m, S_m \cup \{n\}\},\ \text{if } speed_n > 0 \text{ else } \textsc{Sdl}_m\\ \textsc{Sdl}'_n = \textsc{Sched}_{speed_n}\{in_n, out_n, rest_n, U_n, S_n \cup P_\downarrow\}\end{array}}{\begin{array}{c}k[\textsc{Sdl}_k \mid n[\textsc{Sdl}_n \mid \mathbf{in}\ m.P \mid Q] \mid m[\textsc{Sdl}_m \mid R] \mid U]\\ \twoheadrightarrow k[\textsc{Sdl}'_k \mid m[\textsc{Sdl}'_m \mid R \mid n[\textsc{Sdl}'_n \mid P \mid Q]] \mid U]\end{array}} \quad (\text{TR-In})$$

$$\frac{\begin{array}{l}\textsc{Sdl}_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k, S_k\},\quad n \in U_m \cup S_m\\ \textsc{Sdl}_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m, S_m\}\\ \textsc{Sdl}_n = \textsc{Sched}_{speed_n}\{in_n, out_n, rest_n, U_n, S_n\}\\ \textsc{Sdl}'_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k, S_k \cup \{n\}\},\ \text{if } speed_n > 0 \text{ else } \textsc{Sdl}_k\\ \textsc{Sdl}'_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m \setminus \{n\}, S_m \setminus \{n\}\}\\ \textsc{Sdl}'_n = \textsc{Sched}_{speed_n}\{in_n, out_n, rest_n, U_n, S_n \cup P_\downarrow\}\end{array}}{\begin{array}{c}k[\textsc{Sdl}_k \mid m[\textsc{Sdl}_m \mid n[\textsc{Sdl}_n \mid \mathbf{out}\ m.P \mid Q] \mid R] \mid U]\\ \twoheadrightarrow k[\textsc{Sdl}'_k \mid n[\textsc{Sdl}'_n \mid P \mid Q] \mid m[\textsc{Sdl}'_m \mid R] \mid U]\end{array}} \quad (\text{TR-Out})$$

$$\frac{\begin{array}{l}\textsc{Sdl}_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k, S_k\},\quad n \in U_k \cup S_k\\ \textsc{Sdl}'_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k \setminus \{n\}, S_k \setminus \{n\} \cup P_\downarrow \cup R\downarrow\}\end{array}}{k[\textsc{Sdl}_k \mid \mathbf{open}\ n.P \mid n[\textsc{Sdl}_n \mid R] \mid Q] \twoheadrightarrow k[\textsc{Sdl}'_k \mid P \mid R \mid Q]} \quad (\text{TR-Open})$$

$$\frac{\begin{array}{l}\textsc{Sdl}_m = \textsc{Sched}_{speed_k}\{in_m, out_m, rest_m, U_m, S_m\},\quad speed_m > 0\\ \textsc{Sdl}'_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m, S_m \cup \{\mathbf{c}\ .P\}\}\end{array}}{m[\textsc{Sdl}_m \mid \mathbf{c}\ .P \mid R] \twoheadrightarrow m[\textsc{Sdl}'_m \mid R]} \quad (\text{TR-Resource})$$

Table 2. Reduction rules for timed capabilities. A blue backdrop marks the trigger of the reduction, red the changes in the schedulers, and green eventual constraints.

The reduction rules for virtually timed ambients are given in Tables 2 and 3. The timed capabilities **in** n, **out** n, and **open** n enable virtually timed ambients to move in the hierarchical ambient structure. The local schedulers need to know about the current subambients, so their lists of subambients must be adjusted when virtually timed ambients move. Observe that without adjusting the schedulers, the moving subambient would not receive time slices from the scheduler in its new surrounding ambient. In TR-In and TR-Out, the schedulers of the old and new surrounding ambient of the moving ambient are updated by removing and adding, respectively, the name of the moving ambient, if it has

$$\frac{\text{SDL} = \text{SCHED}_{speed}\{in, 0, 0, \emptyset, \emptyset\},\ \ \text{SDL}' = \text{SCHED}_{speed}\{\ in+1\ , 0, 0, \emptyset, \emptyset\},\ R \not\equiv \mathbf{c}\,.P \mid P'}{a[\ \texttt{tick}\ \mid \text{SDL} \mid R] \rightharpoonup a[\text{SDL}' \mid R]} \quad \text{(RR-Empty)}$$

$$\frac{\begin{array}{ll}\text{SDL} = \text{SCHED}_{speed}\{in, 0, 0, U, S\}, & U \cup S \neq \emptyset \\ \text{SDL}' = \text{SCHED}_{speed}\{\ in+1, x, z\ , U, S\}, & speed = x + \sum_{y=1}^{z} \frac{1}{b_y},\ b_y > 1\end{array}}{a[\ \texttt{tick}\ \mid \text{SDL} \mid R] \rightharpoonup a[\text{SDL}' \mid R]} \quad \text{(RR-Tick)}$$

$$\frac{\begin{array}{ll}\text{SDL} = \text{SCHED}_{speed}\{in, out, rest,\ \emptyset, S\ \}, & R \not\equiv \mathbf{c}\,.P \mid P' \\ \text{SDL}' = \text{SCHED}_{speed}\{in, out, rest,\ S, \emptyset\ \} & \end{array}}{a[\text{SDL} \mid R] \rightharpoonup a[\text{SDL}' \mid R]} \quad \text{(RR-NewRound)}$$

$$\frac{\begin{array}{l}\ out > 0\ , a_i \in U,\ a_i \equiv \mathbf{c}\,.P, \qquad \text{SDL} = \text{SCHED}_{speed}\{in, out, rest, U, S\} \\ \text{SDL}' = \text{SCHED}_{speed}\{in,\ out-1\ , rest, U \setminus\ \{a_i\}\ \cup\ P_{\downarrow}\ , S\}\end{array}}{a[\text{SDL} \mid R] \rightharpoonup a[\text{SDL}' \mid R \mid P]} \quad \text{(RR-Tock}_{\text{1-consume}}\text{)}$$

$$\frac{\begin{array}{l}\ out > 0\ , a_i \in U, \qquad R \equiv a_i[\text{SDL}_{a_i} \mid P'] \mid P, \qquad R' \equiv a_i[\text{SDL}_{a_i} \mid\ \texttt{tick}\ \mid P'] \mid P \\ \text{SDL} = \text{SCHED}_{speed}\{in, out, rest, U, S\} \\ \text{SDL}' = \text{SCHED}_{speed}\{in,\ out-1\ , rest, U \setminus\ \{a_i\}\ , S \cup\ \{a_i\}\ \}\end{array}}{a[\text{SDL} \mid R] \rightharpoonup a[\text{SDL}' \mid R']} \quad \text{(RR-Tock}_{\text{1-ambient}}\text{)}$$

$$\frac{\begin{array}{l}\ rest > 0\ , in \bmod b_{rest} = 0,\ a_i \in U,\ a_i \equiv \mathbf{c}\,.P,\ speed = x + \sum_{y=1}^{z} \frac{1}{b_y},\ b_y > 1 \\ \text{SDL} = \text{SCHED}_{speed}\{in, out, rest, U, S\} \\ \text{SDL}' = \text{SCHED}_{speed}\{in, out,\ rest-1\ , U \setminus\ \{a_i\}\ \cup\ P_{\downarrow}\ , S\}\end{array}}{a[\text{SDL} \mid R] \rightharpoonup a[\text{SDL}' \mid R \mid P]} \quad \text{(RR-Tock}_{\text{2-consume}}\text{)}$$

$$\frac{\begin{array}{l}\ rest > 0\ , \quad a_i \in U, \quad R \equiv a_i[\text{SDL}_{a_i} \mid P'] \mid P, \quad R' \equiv a_i[\text{SDL}_{a_i} \mid\ \texttt{tick}\ \mid P'] \mid P \\ \text{SDL} = \text{SCHED}_{speed}\{in, out, rest, U, S\},\ \ in \bmod b_{rest} = 0, \quad speed = x + \sum_{y=1}^{z} \frac{1}{b_y},\ b_y > 1 \\ \text{SDL}' = \text{SCHED}_{speed}\{in, out,\ rest-1\ , U \setminus\ \{a_i\}\ , S \cup\ \{a_i\}\ \}\end{array}}{a[\text{SDL} \mid R] \rightharpoonup a[\text{SDL}' \mid R']} \quad \text{(RR-Tock}_{\text{2-ambient}}\text{)}$$

$$\frac{\begin{array}{l}\ rest > 0\ , in \bmod b_{rest} \neq 0,\ speed = x + \sum_{y=1}^{z} \frac{1}{b_y},\ b_y > 1 \\ \text{SDL} = \text{SCHED}_{speed}\{in, out, rest, U, S\},\ \ \text{SDL}' = \text{SCHED}_{speed}\{in, out,\ rest-1\ , U, S\}\end{array}}{a[\text{SDL} \mid R] \rightharpoonup a[\text{SDL}' \mid R]} \quad \text{(RR-Tock}_{\text{2-no action}}\text{)}$$

$$\frac{\text{SDL}^{\dagger} = \text{SCHED}^{\dagger}\{in, 0, 0, -, U, S\},\ \ \text{SDL}^{\dagger}_{*} = \text{SCHED}^{\dagger}\{\ in+1, 1\ , 0, -, U, S\}}{\text{SDL}^{\dagger} \rightharpoonup\ \text{SDL}^{\dagger}_{*}} \quad \text{(RR-Root)}$$

Table 3. Transition system for fair, preemptive distribution of virtual time slices, where $b_y \in \mathbb{N}$. A blue backdrop marks the reduction trigger and red the changes.

a speed greater than zero. The scheduler of the moving subambient is also updated as it needs to contain the barbs of the process that was hidden behind the movement capability. In TR-Open, the scheduler of the opening ambient itself is updated by removing the name of the opened ambient and adding the barbs of the processes inside this ambient as well as the barbs of the process hidden behind the open capability. The scheduler of the opened ambient is deleted. In TR-Resource, the time consuming process moves into the scheduler, where it awaits the distribution of a time slice as resource before it can continue. This reduction can only happen in virtually timed ambients with speed greater zero, meaning ambients which actually emit resources.

The rules in Table 3 distribute time slices via the local schedulers. We want to enable the schedulers to distribute time slices as soon as possible. The ratio of output time slices to input time slices is defined by the *speed* $\in \mathbb{Q}$ of the scheduler. For example, for a speed of $3/2$ the first incoming time slice (`tick`) should trigger one outgoing time slice and the second input should trigger two, emitting in total three time slices for two inputs. Thus, in order to implement a simple *eager scheduling strategy*, we make use of the so-called *Egyptian fraction decomposition* to determine the number of time slices to be distributed by the local scheduler for each incoming time slice `tick`. For every rational number $q \in \mathbb{Q}$ it holds that $q = x + \sum_{y=1}^{z} \frac{1}{b_y}$ for $x, b_y \in \mathbb{N}$, which is solvable in polynomial time. A greedy algorithm (e.g., [18]) additionally yields the desirable property that a time slice is distributed as soon as possible. From this decomposition, it follows that for each input time slice the local scheduler with speed q will distribute x time slices, plus one additional time slice for every b_y-th input.

In RR-Tick, the local scheduler receives a time slice, which it registers in the counter *in*. At the same time *out* and *rest* initiate the distribution of time slices depending on the Egyptian fraction decomposition of the speed of the scheduler. These steps of the time slice distribution are shown in the RR-Tock rules, which allow transferring a new `tick` to a timed subambient or using the time slice as a resource for a consume capability, which is waiting in the scheduler. The RR-Tock$_1$ rules concern the number x of time slices that are given out for every input time slice, while the RR-Tock$_2$ rules only allow to give out a time slice if the input step is a multiple of one of the fraction denominators b_y. This amounts to a concrete implementation of a fair scheduler where progress is uniform over the queue of timed subambients and time consuming processes. Once all waiting subambients and processes inside the set *unserved* have been served one time slice and are moved to the set *served*, either the rule RR-NewRound ensures that a new round of time slice distribution can begin, or, if the queue is empty, the rule RR-Empty is applied. This scheduling strategy ensures fairness in the competition for resources between processes, as the rounds ensure that no process can bypass another process more than once. The side condition $R \not\equiv \mathbf{c}\,.P \mid P'$ in the rules RR-NewRound and RR-Empty ensures that all resource-consuming processes, which are prefixed by a **c** capability, are included in the set to be scheduled for the next round. The root scheduler Sched† reduces without time slices from surrounding ambients in RR-Root.

In the sequel we will focus on a subset of the language of virtually timed ambients without replication and without restriction, denoted by VTA^-. Similarly, let MA^- denote mobile ambients without replication and without restriction.

Example 1 (Virtually timed subambients, scheduling and resource consumption). The virtually timed ambient *cloud*, exemplifying a cloud server, emits one time slice for every time slice it receives, $\text{SDL}_{cloud} = \text{SCHED}_1\{0,0,0,\emptyset,\emptyset\}$. It contains two `tick` and is entered by a virtually timed subambient *vm*.

$$\begin{aligned} &cloud\,[\text{SCHED}_1\{0,0,0,\emptyset,\emptyset\} \mid \texttt{tick} \mid \texttt{tick}] \\ &\mid vm[\text{SCHED}_{3/4}\{0,0,0,\emptyset,\emptyset\} \mid \mathbf{in}\ cloud.\ \mathbf{c}\ .P] \end{aligned}$$

The ambient *vm* exemplifies a virtual machine containing a resource consuming task, where $\text{SDL}_{vm} = \text{SCHED}_{3/4}\{0,0,0,\emptyset,\emptyset\}$. The Egyptian fraction decomposition of the speed yields $3/4 = 0 + 1/2 + 1/4$ meaning that there is no time slice given out for every incoming time slice, but one time slice for every second incoming time slice, and one for every fourth. The process reduces as follows:

$$\begin{aligned}
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{0,0,0,\emptyset,vm\} \mid \texttt{tick} \mid \texttt{tick} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\emptyset,\emptyset\} \mid \mathbf{c}\ .P]] && (\text{TR-IN}) \\
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{0,0,0,vm,\emptyset\} \mid \texttt{tick} \mid \texttt{tick} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\emptyset,\emptyset\} \mid \mathbf{c}\ .P]] && (\text{RR-NEWROUND}) \\
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{0,0,0,vm,\emptyset\} \mid \texttt{tick} \mid \texttt{tick} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\emptyset,\mathbf{c}\ .P\} \mid \mathbf{0}]] && (\text{TR-RESOURCE}) \\
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{0,0,0,vm,\emptyset\} \mid \texttt{tick} \mid \texttt{tick} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\mathbf{c}\ .P,\emptyset\} \mid \mathbf{0}]] && (\text{RR-NEWROUND}).
\end{aligned}$$

Here, the ambient *vm* enters the ambient *cloud* and is registered in the scheduler. Furthermore, the resource consuming process in *vm* is registered. In the next steps the time slices move into the scheduler of the *cloud* ambient and are distributed further down in the hierarchy.

$$\begin{aligned}
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{1,1,0,vm,\emptyset\} \mid \texttt{tick} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\mathbf{c}\ .P,\emptyset\} \mid \mathbf{0}]] && (\text{RR-TICK}) \\
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{1,0,0,\emptyset,vm\} \mid \texttt{tick} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\mathbf{c}\ .P,\emptyset\} \mid \texttt{tick}]] && (\text{RR-TOCK}_{\text{1-AMBIENT}}) \\
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{2,0,0,vm,\emptyset\} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\mathbf{c}\ .P,\emptyset\} \mid \texttt{tick}]] && (\text{RR-NEWROUND}) \\
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{2,1,0,vm,\emptyset\} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\mathbf{c}\ .P,\emptyset\} \mid \texttt{tick}]] && (\text{RR-TICK}) \\
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{2,0,0,\emptyset,vm\} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\mathbf{c}\ .P,\emptyset\} \mid \texttt{tick} \mid \texttt{tick}]] && (\text{RR-TOCK}_{\text{1-AMBIENT}}) \\
&\rightarrowtriangle cloud\,[\text{SCHED}_1\{2,0,0,vm,\emptyset\} \\
&\quad \mid vm[\text{SCHED}_{3/4}\{0,0,0,\mathbf{c}\ .P,\emptyset\} \mid \texttt{tick} \mid \texttt{tick}]] && (\text{RR-NEWROUND}).
\end{aligned}$$

$\mathcal{A}, B ::=$	TRUE	true
	$\neg\mathcal{A}$	negation
	$\mathcal{A} \vee \mathcal{B}$	disjunction
	$\mathbf{0}$	void
	$n[\mathcal{A}]$	location
	$\mathcal{A} \mid \mathcal{B}$	composition
	$\forall n.\mathcal{A}$	universal quantification over names
	$\mathcal{A}@n$	local adjunct
	$\mathfrak{c}$	consumption
	$\diamond_{x@n}\mathcal{A}$	sometime modality
	$\diamondsuit_{(speed,s)}\mathcal{A}$	somewhere modality

Table 4. Logical formulas, $n \in$ names, $x, s \in \mathbb{N}_0 \cup \{\infty\}$, $speed \in \mathbb{Q}$

Now the ambient vm can use the time signals to enable resource consumption.

$$
\begin{array}{ll}
\rightarrow cloud\,[\text{SCHED}_1\{2,0,0,vm,\emptyset\} & \\
\quad \mid vm[\text{SCHED}_{3/4}\{1,0,1,\mathbf{c}\,.P,\emptyset\} \mid \texttt{tick}]] & (\text{RR-TICK}) \\
\rightarrow cloud\,[\text{SCHED}_1\{2,0,0,vm,\emptyset\} & \\
\quad \mid vm[\text{SCHED}_{3/4}\{1,0,0,\mathbf{c}\,.P,\emptyset\} \mid \texttt{tick}]] & (\text{RR-TOCK}_{\text{2-NO ACTION}}) \\
\rightarrow cloud\,[\text{SCHED}_1\{2,0,0,vm,\emptyset\} & \\
\quad \mid vm[\text{SCHED}_{3/4}\{2,0,1,\mathbf{c}\,.P,\emptyset\} \mid \mathbf{0}]] & (\text{RR-TICK}) \\
\rightarrow cloud\,[\text{SCHED}_1\{2,0,0,vm,\emptyset\} & \\
\quad \mid vm[\text{SCHED}_{3/4}\{2,0,0,P_{\downarrow},\emptyset\} \mid P]] & (\text{RR-TOCK}_{\text{2-CONSUME}})
\end{array}
$$

Note that, as the calculus is non-deterministic, the reduction rules can be applied in arbitrary order, making several reduction paths possible.

3 Modal Logic for Virtually Timed Ambients

To capture the distinctive features of virtual time and resource provisioning in virtually timed ambients, the modal logic $\mathcal{ML}_{VTA}$ for VTA$^-$ combines the modal logic $\mathcal{ML}_{MA}$ for mobile ambients without the composition adjunct, with notions based on metric temporal logic [24,31,32].

The syntax of $\mathcal{ML}_{VTA}$ is shown in Table 4. The *sometime* operator (the name refers to *sometime in the reduction*) comes with a constraint giving the *maximal number of resources* $x \in \mathbb{N}_0 \cup \{\infty\}$ that a process may use inside an ambient named n before fulfilling formula $\mathcal{A}$. The *somewhere* operator refers to the formula being true in a sublocation of the process and specifies the *minimal speed* that the sublocation must possess relative to its surrounding ambients as well as the *maximal number of subambients* in this location. To define these operators, we adapt the sublocation relation from [9] to accommodate schedulers.

Definition 5 (Sublocation with schedulers). *A process P' is a sublocation of P, written $P \downarrow P'$, iff $P \equiv (n[\text{SDL} \mid P'] \mid P'')$ for some name n, scheduler* SDL*, and process P''. Let $P \downarrow^* P'$ denote the reflexive and transitive closure of $P \downarrow P'$; i.e., $P \downarrow^* P'$ iff $P \downarrow P'$ or $P \downarrow P''$ and $P'' \downarrow^* P'$ for some process P''.*

In order to capture the number of resources consumed in a given ambient, we define a *labeled reduction relation.* While $\twoheadrightarrow$ refers to all reduction steps in virtually timed ambients, we denote by $\xrightarrow{\texttt{tick}}$ the steps of the (RR-TICK) and (RR-EMPTY) rules; i.e., these labeled transitions capture the internal reductions in the schedulers enabling the timed reduction of processes. All other reduction steps are marked by $\xrightarrow{\tau}$.

Definition 6. *$P \xrightarrow{\texttt{tick}} P'$ iff $P \mid \texttt{tick} \to P'$. We write $\xrightarrow{\texttt{tick}^x}$ if x time signals* tick *are used; i.e., $P \xrightarrow{\texttt{tick}^x} P'$ iff $P \mid \texttt{tick} \mid \cdots \mid \texttt{tick} \to^* P'$, where the number of time signals* tick *is x. The weak version of this reduction is defined as $P \xRightarrow{\texttt{tick}^x} P'$ iff $P(\xrightarrow{\tau}^* \xrightarrow{\texttt{tick}} \xrightarrow{\tau}^*)^x P'$, where $\xrightarrow{\tau}^*$ describes the application of an arbitrary number of τ-steps.*

The relation $\xRightarrow{\texttt{tick}^x}_n$ captures the number of resources used inside an ambient n inside a process.

Definition 7. *$P \xRightarrow{\texttt{tick}^x}_n P'$ iff $P \twoheadrightarrow^* P'$ and there exists Q, Q' such that $P \downarrow^* n[Q]$, $P' \downarrow^* n[Q']$ and $Q \xRightarrow{\texttt{tick}^x} Q'$.*

We now define the notion of *accumulated speed*, based on the eager distribution strategy for time slices. The accumulated speed $accum\{m\}_P \in \mathbb{Q}$ in a subambient m which is part of a process P, is the relative speed of the ambient with respect to the scheduler of P and the siblings.

Definition 8 (Accumulated speed). *Let $speed_k \in \mathbb{Q}$ and $children(k)$ denote the speed and number of children of a virtually timed ambient* k*. Let* m *be a timed subambient of a process P, the name* parent *denoting the direct parental ambient of* m*, and* C *the path of all parental ambients of* m *up to the level of P. The* accumulated speed *for preemptive scheduling in a subambient* m *up to the level of the process P is given by*

$$\begin{aligned} accum\{m\}_P &= speed_m \cdot {}^1/_{children(parent)} \cdot speed_{parent} \\ &= speed_m \cdot \prod_{k \in C} {}^1/_{children(k)} \cdot \prod_{k \in C} speed_k \end{aligned}$$

Schedulers distribute time slices preemptively, as child processes get one time slice at the time in iterative rounds. Consequently, an ambient's accumulated speed is influenced by both the speed and the number of children n of the parental ambient. Thus, scheduling is not only *path sensitive* but also *sibling sensitive.*

The satisfaction relation for logical formula, defined inductively in Table 5, can now be explained using these definitions. A process P satisfies the *negation* of a formula $\mathcal{A}$ iff P does not satisfy $\mathcal{A}$. The *disjunction* $\mathcal{A} \vee \mathcal{B}$ is satisfied by a process which satisfies either $\mathcal{A}$ or $\mathcal{B}$. A process satisfies the formula $\mathbf{0}$ (*void*) iff the process is equivalent to the inactive process $\mathbf{0}$. A process P satisfies a formula $\mathcal{A}$ *in location* n iff P is equivalent to $n[P']$ and P' satisfies $\mathcal{A}$. The *composition*

$P \vDash \text{TRUE}$		
$P \vDash \neg\mathcal{A}$	iff	$P \nvDash \mathcal{A}$
$P \vDash \mathcal{A} \vee \mathcal{B}$	iff	$P \vDash \mathcal{A} \vee P \vDash \mathcal{B}$
$P \vDash \mathbf{0}$	iff	$P \equiv \mathbf{0}$
$P \vDash n[\mathcal{A}]$	iff	$\exists P'$ s.t. $P \equiv n[P'] \wedge P' \vDash \mathcal{A}$
$P \vDash \mathcal{A} \mid \mathcal{B}$	iff	$\exists P', P''$ s.t. $P \equiv P' \mid P'' \wedge P' \vDash \mathcal{A} \wedge P'' \vDash \mathcal{B}$
$P \vDash \forall n.\mathcal{A}$	iff	$\forall m : P \vDash \mathcal{A}\{n \leftarrow m\}$
$P \vDash \mathcal{A}@n$	iff	$n[P] \vDash \mathcal{A}$
$P \vDash \mathfrak{c}$	iff	$\exists P', P'', P'''$ s.t. $P \equiv P'.\, \mathbf{c}\, .P'' \mid P''' \vee P \downarrow^* (P'.\, \mathbf{c}\, .P'' \mid P''')$
$P \vDash \diamond_{x@n}\mathcal{A}$	iff	$\exists P'$ s.t. $P \xRightarrow{\mathtt{tick}^y}_n P' \wedge y \leq x \wedge P' \vDash \mathcal{A}$
$P \vDash \Diamond_{(speed,s)}\mathcal{A}$	iff	$\exists P', P'', n$ s.t. $(P \equiv n[\text{SDL} \mid P'] \mid P'' \vee P \downarrow^* n[\text{SDL} \mid P'])$ $\wedge P' \vDash \mathcal{A} \wedge accum\{n\}_P \geq speed \wedge \lvert U_{\text{SDL}} \cup S_{\text{SDL}} \rvert \leq s$

Table 5. Satisfaction of logical formulas, $n \in$ names, $x, s \in \mathbb{N}_0 \cup \{\infty\}$, $speed \in \mathbb{Q}$

$\mathcal{A} \mid \mathcal{B}$ is satisfied by a process iff the process can be split into two parallel processes, such that one satisfies $\mathcal{A}$ and the other $\mathcal{B}$. *Universal quantification* $\forall n.\mathcal{A}$ over names is satisfied iff $\mathcal{A}$ holds for all names n. A process satisfies the *local adjunct* iff it satisfies the formula $\mathcal{A}$ in location n. The *consumption* formula $\mathfrak{c}$ is satisfied by any process which contains a consumption capability. A process P satisfies the *sometime modality* iff it reduces to a process satisfying the formula, and uses less than x resources in ambient n in the reduction. The *somewhere modality* is satisfied iff there exists a sublocation of P satisfying the formula, the relative speed in the sublocation is greater or equal to the given *speed*, and the sublocation has less than or equal to s timed subambients.

We show that $\mathcal{ML}_{VTA}$ is conservative with respect to $\mathcal{ML}_{MA}$. It holds that every process in mobile ambients has an equivalent process in virtually timed ambients when timing aspects are ignored. We attach the names of the logics to the satisfaction relation to distinguish the relations in the presentation.

Lemma 1 (Correspondence to untimed processes). *Let $\mathcal{A} \in \mathcal{ML}_{MA}$ and $P \in MA^-$. If $P \vDash_{\mathcal{ML}_{MA}} \mathcal{A}$ then there exists $P' \in VTA^-$ such that $P' \vDash_{\mathcal{ML}_{VTA}} \mathcal{A}$.*

The satisfaction relation for the untimed definitions of the sometime and somewhere modalities in $\mathcal{ML}_{MA}$ is given as:

$$P \vDash_{\mathcal{ML}_{MA}} \diamond\mathcal{A} \iff \exists P' \text{ s.t. } P \rightarrow^* P' \wedge P' \vDash_{\mathcal{ML}_{MA}} \mathcal{A}$$
$$P \vDash_{\mathcal{ML}_{MA}} \Diamond\mathcal{A} \iff \exists P' \text{ s.t. } P \downarrow^* P' \wedge P' \vDash_{\mathcal{ML}_{MA}} \mathcal{A}.$$

These definitions correspond to timed modalities without restrictions on names and resources.

Lemma 2 (Correspondence to untimed modalities). *For all $P \in VTA^-$, $\mathcal{A} \in \mathcal{ML}_{MA}$ it holds that*

1. $P \vDash_{\mathcal{ML}_{MA}} \diamond\mathcal{A} \iff P \vDash_{\mathcal{ML}_{VTA}} \neg\forall n \neg(\diamond_{\infty @ n}\mathcal{A})$
2. $P \vDash_{\mathcal{ML}_{MA}} \Diamond\mathcal{A} \iff P \vDash_{\mathcal{ML}_{VTA}} \Diamond_{(0,\infty)}\mathcal{A}.$

Proof. Follows from the definition of the satisfaction relation.

1. $P \vDash_{\mathcal{ML}_{VTA}} \neg \forall n \neg (\diamond_{\infty @ n} \mathcal{A})$

$$\iff P \nvDash_{\mathcal{ML}_{VTA}} \forall n \neg (\diamond_{\infty @ n} \mathcal{A})$$
$$\iff \forall m : P \nvDash_{\mathcal{ML}_{VTA}} \neg (\diamond_{\infty @ n} \mathcal{A})\{n \leftarrow m\}$$
$$\iff P \nvDash_{\mathcal{ML}_{VTA}} \neg (\diamond_{\infty @ m_1} \mathcal{A}) \wedge \cdots \wedge \neg (\diamond_{\infty @ m_k} \mathcal{A})$$
$$\iff P \vDash_{\mathcal{ML}_{VTA}} \diamond_{\infty @ m_i} \mathcal{A}, \text{ for any } m_i$$
$$\iff \exists P' \text{ s.t. } P \xRightarrow{\texttt{tick}^y}_{m_i} P' \wedge y \leq \infty \wedge P' \vDash_{\mathcal{ML}_{MA}} \mathcal{A}, \text{ for any } m_i$$
$$\iff \exists P' \text{ s.t. } P \twoheadrightarrow^* P' \wedge P' \vDash_{\mathcal{ML}_{MA}} \mathcal{A}$$
$$\iff P \vDash_{\mathcal{ML}_{MA}} \diamond \mathcal{A}$$

2. $P \vDash_{\mathcal{ML}_{VTA}} \Diamond_{(0,\infty)} \mathcal{A}$

$$\iff \exists P', P'', n \text{ s.t. } (P \equiv n[\textsc{Sdl} \mid P'] \mid P'' \vee P \downarrow^* n[\textsc{Sdl} \mid P'])$$
$$\wedge\, P' \vDash \mathcal{A} \wedge \mathit{accum}\{n\}_P \geq 0 \wedge |U_{\textsc{Sdl}} \cup S_{\textsc{Sdl}}| \leq \infty$$
$$\iff \exists P' \text{ s.t. } P \downarrow^* P' \wedge P' \vDash_{\mathcal{ML}_{MA}} \mathcal{A}$$
$$\iff P \vDash_{\mathcal{ML}_{MA}} \Diamond \mathcal{A}$$

For all other cases, the definition of the satisfaction relation in $\mathcal{ML}_{MA}$ is the same as in $\mathcal{ML}_{VTA}$. Thus, we can translate a $\mathcal{ML}_{MA}$-formula to $\mathcal{ML}_{VTA}$ by substituting untimed with timed modalities as given above. We now prove that $\mathcal{ML}_{VTA}$ is a conservative extension of $\mathcal{ML}_{MA}$.

Theorem 1 (Conservative extension). *Let $A \in \mathcal{ML}_{MA}$ and $P \in MA^-$. If $P \vDash_{\mathcal{ML}_{MA}} \mathcal{A}$ then there exists $P' \in VTA^-$ such that $P' \vDash_{\mathcal{ML}_{VTA}} \mathcal{A}^*$, where $\mathcal{A}^*$ is the translation of $\mathcal{A}$ to $\mathcal{ML}_{VTA}$.*

Proof. Follows from Lemmas 1 and 2 and the fact that for all other cases than the modalities, the satisfaction relation in $\mathcal{ML}_{MA}$ stays the same in $\mathcal{ML}_{VTA}$.

Example 2 (Modal contracts for virtually timed processes). Let process P consist of a *cloud* ambient containing a virtual machine *vm*, similar to Example 1, and a *task* to enter *vm* in order to consume a resource:

$$P \equiv \mathit{cloud}[\textsc{Sdl}_{cloud} \mid \texttt{tick} \mid \texttt{tick} \mid \mathit{vm}[\textsc{Sdl}_{vm} \mid \mathbf{open}\ \mathit{task}] \mid \mathit{task}[\mathbf{in}\ \mathit{vm}.\ \mathbf{c}]].$$

This system satisfies the modal contract given by the formula $\diamond_{2@vm}(\neg \mathfrak{c})$, which expresses that after using two time slices the task can be executed. Example 1 illustrates how the time slices move from the *cloud* ambient into the virtual machine. Afterwards we can observe the following reduction process inside the *cloud* ambient:

$$\mathit{vm}[\textsc{Sdl}_{vm} \mid \texttt{tick} \mid \texttt{tick} \mid \mathbf{open}\ \mathit{task}] \mid \mathit{task}[\mathbf{in}\ \mathit{vm}.\ \mathbf{c}]$$
$$\twoheadrightarrow \mathit{vm}[\textsc{Sdl}_{vm} \mid \texttt{tick} \mid \texttt{tick} \mid \mathbf{open}\ \mathit{task} \mid \mathit{task}[\mathbf{c}]]$$
$$\twoheadrightarrow \mathit{vm}[\textsc{Sdl}_{vm} \mid \texttt{tick} \mid \texttt{tick} \mid \mathbf{c}]$$
$$\twoheadrightarrow \mathit{vm}[\textsc{Sdl}_{vm} \mid \mathbf{0}]$$

This shows that $P \vDash \diamond_{2@vm}(\neg \mathfrak{c})$. With two time signals from the original active level the task can be executed. Therefore, we can say that P satisfies the modal contract stating that the system is able to execute with the use of two resources.

4 A Model Checker for Virtually Timed Ambients

To answer the question whether a process in VTA$^-$ satisfies a given formula, we create a model checker algorithm for $\mathcal{ML}_{VTA}$. We extend the model checker algorithm for $\mathcal{ML}_{MA}$ [9] to cover the properties of virtually timed ambients. Technically, we add $\mathbf{c}\ .P$ and $\mathtt{tick}$ to the *prime processes* and use the same notion of *normal form*, where we add $Norm(n[\textsc{Sdl} \mid P]) \triangleq [n[\textsc{Sdl} \mid P]]$. Furthermore, the *Reachable* and *SubLocations* routines must account for our changes to the *sometime* and *somewhere* modalities and a *Consumption* routine is added to check if the formula $\mathfrak{c}$ holds for a process. These routines are now defined for $\mathcal{ML}_{VTA}$.

Definition 9. *Let $P \in VTA^-$, then*

- *$Reachable^x_n(P) = [P_1, \ldots, P_k]$ iff $P \xrightarrow{\mathtt{tick}^y}_n P_i$, for all $i \in 1, \ldots, k$, $y \leq x$ and for all Q, if $P \xrightarrow{\mathtt{tick}^y}_n Q$ then $Q \equiv P_i$ for some $i \in 1, \ldots, k$ and $y \leq x$.*
- *$SubLocations_{(speed,s)}(P) = [P_1, \ldots, P_k]$ iff $P \equiv n[\textsc{Sdl} \mid P_i] \mid P'$ or $P \downarrow^* n[\textsc{Sdl} \mid P_i]$ for some n and $accum\{n\}_P \geq speed$ and $|S_{\textsc{Sdl}_n} \cup T_{\textsc{Sdl}_n}| \leq s$, for all $i \in 1, \ldots, k$. And for all Q, if $P \equiv n[\textsc{Sdl} \mid Q \mid P'$ or $P \downarrow^* n[\textsc{Sdl} \mid Q]$ some n and $accum\{n\}_P \geq speed$ and $|S_{\textsc{Sdl}_n} \cup T_{\textsc{Sdl}_n}| \leq s$, then $Q \equiv P_i$ for some $i \in 1, \ldots, k$.*
- *$Consumption(P) = \textsc{True}$ iff $SubLocations_{(0,\infty)}(P) = [P_1, \ldots, P_k]$ and $\exists P'$, P'', $P''', P_i, i \in 1 \ldots k$ such that $P_i \equiv P'.\ \mathbf{c}\ .P'' \mid P'''$.*

The model checker algorithm for $\mathcal{ML}_{VTA}$ is defined inductively as follows:

Check$(P, \mathcal{A})$: Checking whether process P satisfies formula $\mathcal{A}$

$Check(P, \textsc{True}) \triangleq \textsc{True}$
$Check(P, \neg\mathcal{A}) \triangleq \neg Check(P, \mathcal{A})$
$Check(P, \mathcal{A} \vee \mathcal{B}) \triangleq Check(P, \mathcal{A}) \vee Check(P, \mathcal{B})$
$Check(P, \mathbf{0}) \triangleq$ if $Norm(P) = []$ then $\textsc{True}$ else $\textsc{False}$.
$Check(P, n[\mathcal{A}]) \triangleq$ if $Norm(P) = n[Q]$ for some Q then $Check(Q, \mathcal{A})$ else $\textsc{False}$.
$Check(P, \mathcal{A} \mid \mathcal{B}) \triangleq$ Let $Norm(P) = [\pi_1, \ldots, \pi_k]$:
  $\exists I, J$ s.t. $I \cup J = \{1, \ldots, k\}$ and $I \cap J = \emptyset$:
  $\bigvee_{I,J} Check(\prod_{i \in I} \pi_i, \mathcal{A}) \wedge Check(\prod_{j \in J} \pi_j, \mathcal{B})$
$Check(P, \forall n.\mathcal{A}) \triangleq$ Let $\{m_1, \ldots, m_k\} = fn(P) \cup fn(\mathcal{A})$ and $m_0 \notin \{m_1, \ldots, m_k\}$:
  $\bigwedge_{i \in 0 \ldots k} Check(P, \mathcal{A}\{n \leftarrow m_i\})$
$Check(P, \mathfrak{c}) \triangleq Consumption(P)$
$Check(P, \diamond_{x@n}\mathcal{A}) \triangleq$ Let $Reachable^x_n(P) = [P_1, \ldots, P_k]$:
  $\bigvee_{i \in 1, \ldots, k} Check(P_i, \mathcal{A})$
$Check(P, \diamondsuit_{(speed,s)}\mathcal{A}) \triangleq$ Let $SubLocations_{(speed,s)}(P) = [P_1, \ldots, P_k]$:
  $\bigvee_{i \in 1, \ldots, k} Check(P_i, \mathcal{A})$
$Check(P, \mathcal{A}@n) \triangleq Check(n[P], \mathcal{A})$

As our extension only adds the simple predicate $\mathfrak{c}$ to the model checker and imposes discreet restrictions on the *Reachable* and *SubLocations* properties, it follows from results in [9] and [14] (regarding the equivalence of processes and their norms) that all recursive calls of the algorithm are on subformulas, therefore the algorithm always terminates.

Theorem 2. *For $P \in VTA^-$, $\mathcal{A} \in \mathcal{ML}_{VTA}$ it holds that:*

$$P \vDash \mathcal{A} \textit{ iff } \mathit{Check}(P, \mathcal{A}) = \textsc{True}.$$

Example 3 (Model checking). Reconsider Example 2, where the satisfaction of the sometime formula was demonstrated by considering the reduction. Let $P = vm[\textsc{Sdl}_{vm} \mid \texttt{tick} \mid \texttt{tick} \mid \textbf{open}\ task] \mid task[\textbf{in}\ vm.\ \mathbf{c}]$. We will now show that

$$\mathit{Check}(P, \diamond_{2@vm}(\neg\mathfrak{c})) = \textsc{True}.$$

It holds that

$$\mathit{Check}(P, \diamond_{2@vm}(\neg\mathfrak{c})) \triangleq \text{Let } \mathit{Reachable}^2_{vm}(P) = [P_1, \ldots, P_k] : \bigvee_{i \in 1,\ldots,k} \mathit{Check}(P_i, \neg\mathfrak{c})$$

$\mathit{Reachable}^2_{vm}(P)$ contains all states reachable from P with two timed steps and arbitrary many τ-steps. This includes $P_j = vm[\textsc{Sdl}_{vm} \mid \mathbf{0}]$. For this process it holds that $\mathit{Check}(P_j, \neg\mathfrak{c}) \triangleq \neg\mathit{Check}(P_j, \mathfrak{c})$ and $\mathit{Check}(P_j, \mathfrak{c}) \triangleq \mathit{Consumption}(P_j)$ As $\mathit{Consumption}(P_j) = \textsc{False}$ it follows that $\mathit{Check}(P, \diamond_{2@vm}(\neg\mathfrak{c})) = \textsc{True}$.

5 Implementation in Maude

We implement a model checker for $\mathcal{ML}_{VTA}$ in the Maude [16,30] rewriting logic system. Rewriting logic is a flexible, executable formal notation which can be used to represent a wide range of systems and logics with *low representational distance* [26]. Rewriting logic embeds *membership equational logic*, such that a specification or program may contain both equations and rewrite rules. When executing a Maude specification, rewrite steps are applied to normal forms in the equational logic. (The Maude system assumes that the equation set is terminating and confluent.) Thus, equations and rewrite rules constitute the *statics* and *dynamics* of a specification, respectively. Both equations and rewrite rules may be *conditional*, meaning that specified conditions must hold for the rule or equation to apply.

A translation of mobile ambients to Maude was proposed in [34], motivated by the application of the analysis tools that come with the Maude system. However, our primary goal is to build a model checker for virtually timed ambients. Hence, our implementation[1] consists of a translation for VTA$^-$ and $\mathcal{ML}_{VTA}$ to Maude, and will use the Maude engine as the model checker.

The syntax of VTA$^-$, given in Table 1, is represented by Maude terms, constructed from operators:

[1] The full source code is available at `https://github.com/larstvei/Check-VTA/tree/modal-contracts`

```
op zero : -> VTA [ctor] .
op _|_ : VTA VTA -> VTA [id: zero assoc comm] .
op _._ : Capability VTA -> VTA .
op _[_|_] : Name Scheduler VTA -> VTA .
```

The correlation between the formal definition and the Maude specification should be clear. The operator `zero` represents the inactive process, and parallel composition has the algebraic properties of being associative, commutative and having `zero` as identity element. Capability prefixing is represented with a dot. Virtually timed ambients are represented with a name followed by brackets, containing a scheduler and a process. Here all processes are defined with the data type `VTA`. The sort declarations for `VTA`, `Capability`, `Name` and `Scheduler`, as well as syntax for names and capabilities, are omitted.

The reduction rules for timed capabilities (Table 2) are represented as *rewrite rules*, which express that any term or subterm which matches the left hand side of the rewrite relation `=>` may be rewritten into the right hand side; this corresponds to the reduction relation $\rightharpoondown$ in the calculus. Preconditions are expressed using conditional rewrite rules, where a condition is given after the keyword `if`. The TR-IN rule, for instance, may be expressed in Maude as follows:

```
crl [in] :
    K[sched SpdK {InK, OutK, RestK, UnSrvK, SrvK}
      | N[sched SpdN {InN, OutN, RestN, SrvN, UnSrvN} | in(M) . P | Q]
      | M[sched SpdM {InM, OutM, RestM, SrvM, UnSrvM} | R] | U]
  =>
    K[sched SpdK {InK, OutK, RestK, (UnSrvK \ N), (SrvK \ N)}
      | M[sched SpdM {InM, OutM, RestM, SrvM, union(UnSrvM, N)} | R
        | N[sched SpdN {InN, OutN, RestN, SrvN, union(UnSrvN, barb(P))}
          | P | Q]] | U]
  if N in union(UnSrvK, SrvK) .
```

The model checker algorithm *Check* (from Section 3) uses a normal form. Since rule matching in Maude is modulo associativity, commutativity and identity (so-called ACI-matching [16]), the satisfiability conditions of the modal logic can be represented directly, without this normal form. This results in a compact and flexible model checker which stays close to its mathematical formulation.

Terms representing logical formulas (defined in Table 4) are built from operator declarations in Maude and variable substitution on formulas is formalized using recursive equations. The semantics of formulas is interpreted with regards to the calculus of virtually timed ambients, and is formalized by defining the satisfaction relation as an operator:

```
op _|=_ : VTA Formula -> Bool [frozen] .
```

Here, the operator declaration's `frozen` attribute prevents the subterms of a satisfaction formula from being rewritten, giving the model checker control over the rewriting (i.e., the `frozen` attribute prohibits subterm matching). The semantics of the satisfaction relations from Table 5 is expressed as a set of equations and a single rewrite rule. For formulas which only depend on the current state of the process, the satisfaction predicate can be defined by an equation in Maude. For example, negation is defined as follows:

```
eq [Negation] : P |= ~ F = not (P |= F) .
```

Parallel composition relies on the matching of parallel processes, and there may be several possible solutions. Therefore, the satisfaction predicate for parallel processes must be defined as a rule. The rule uses reachability predicates as conditions, which allows the Maude implementation to closely reflect the satisfaction relation.

```
crl [Parallel] : P | Q |= F | G => true
    if P |= F => true /\ Q |= G => true
```

The *sometime modality* constructs formulas that depend on how a process evolves over time. The following conditional rewrite rule captures the semantics of a sometime formula:

```
crl [Sometime] : P |= <> A @ N F => true
  if contains(P, N) /\
     P => Q /\
     distance(P, Q, N) ≤ A /\
     contains(Q, N) /\
     Q |= F => true .
```

In this rule, the terms `contains` and `distance` define the existence of the name in the given process and the number of used resources, and are reduced by equations. Similar to the conditions of the `Parallel` rule, the condition `P => Q` expresses that the pattern `Q` is reachable from a pattern `P` (after substitution in the matching) by the rewrite relation `=>` in one or more steps. Maude will search for a `Q` such that the condition holds using a breadth-first strategy. This useful feature of Maude enables a straightforward implementation of the sometime modality. Note that `Q |= F => true` is used in favor of the simpler `Q |= F` to support nested modal formulas.

The execution of rewrite rules is represented in the syntax of the Maude model checker by providing the rewriting command `rewrite` with satisfaction relation containing a virtually timed ambient and a formula. The resulting Maude program can easily be used to check modal properties for virtually timed ambients and is demonstrated in the following example. The `rewrite` command applies the defined rewrite rules to the given satisfaction relation until termination, at which point the model checker returns a `result` in the form of a `Bool`.

Example 4 (Implementation of modal contracts for virtually timed processes). To illustrate the model checker we implement Example 2. A root ambient contains a virtual machine, which is entered by a request. We check if the system satisfies the quality of service contract stating that the request can be executed after the use of two time slices. The model checker confirms that after the use of two time signals in the *root* ambient there is no consume capability left, meaning that there exists a reduction path where at most two time signals are needed to execute the request in the virtual machine.

6 Related Work

Virtually timed ambients are based on mobile ambients [10]. The calculus is first described in [22]. Mobile ambients model both location mobility and nested locations, and capture processes executing at distributed locations in networks such as the Internet. Gordon proposed a simple formalism for virtualization loosely based on mobile ambients in [20]. The calculus of virtually timed ambients [22,23] stays closer to the syntax of the original mobile ambient calculus, while at the same time including notions of *time* and *explicit resource provisioning*.

Timed process algebras which originated from ACP and CSP can be found in, e.g., [5,29,6]. As virtually timed ambients build upon mobile ambients, we focus the discussion of related work on the π-calculus [35], which originated from CCS and is closely related to the ambient calculus. Timers have been studied for both the distributed π-calculus [8,33] and for mobile ambients [4,3,15]. In this line of work, timers, which are introduced to express the possibility of a timeout, are controlled by a global clock. In contrast, the root schedulers in our work recursively control local schedulers which define the execution power of the nested virtually timed ambients. Modeling timeouts is a straightforward extension of our work.

Modal logic for mobile ambients was introduced to describe properties of spatial configuration and mobile computation [9] for a fragment of mobile ambients without replication and restriction on names, and features a model checker algorithm for the given language fragment and modal logic using techniques from [12] to establish the $\mathit{Reachable}(P)$ and $\mathit{SubLocation}(P)$ properties. The complexity of model checking for mobile ambients is investigated in [13], and shown to be PSPACE-complete. After Cardelli and Gordon's work on logical properties for name restriction [11], the model checker algorithm was extended for private names [14] while preserving decidability and the complexity of the original fragment. Further it was shown that it is not possible to extend the algorithm for replication in the calculus or the local adjunct in the logic, as either of these extensions would lead to undecidability. For simplicity, we base our logic and model checker on the original fragment from [9]. The modal operators with restrictions on timing in this paper borrows ideas from metric temporal logic [24,31,32].

The Process Analysis Toolkit (PAT) [36] has been used to specify processes in the ambient calculus as well as properties in modal logic [37], to provide a basis for a possible model checker implementation. A model checker for ambient logic has been implemented by separating the analysis of temporal and spatial properties [2]: Mobile ambients are translated into Kripke structures and spatial modalities are replaced with atomic propositions in order to reduce ambient logic formulas to temporal logic formulas, while the analysis of temporal modalities are handled using the NuSMV model checker. In contrast to our work, none of the above model checkers consider notions of time or resources. We use Maude [16] to implement our model checker, exploiting the low representational distance which distinguishes this system [26]. The operational reduction rules for mobile ambients as well as a type system have been implemented in Maude in [34]. In contrast, our implementation focuses on capturing the timed reduction rules of virtually timed ambients as well as the modal formulas to define a model checker.

7 Concluding Remarks

Virtualization opens for new and interesting formal computational models. This paper introduces modal contracts to capture quality of service properties for virtually timed ambients, a formal model of hierarchical locations of execution. Resource provisioning for virtually timed ambients is based on virtual time, a local notion of time reminiscent of time slices for virtual machines in the context of nested virtualization. These time slices are locally distributed by means of fair, preemptive scheduling. Modal contracts are formalized as propositions in a modal logic for virtually timed ambients which features notions from metric temporal logic, enabling the timed behavior and resource consumption of a system to be expressed as modal logic properties of processes. We can now prove whether a system satisfies a certain quality of service agreement captured as a modal contract by means of a model checking algorithm which proves that a process satisfies a formula. We provide a proof of concept implementation of the model checking algorithm in the Maude rewriting logic system.

To model active resource management, future work will extend the model with constructs to support resource-aware scaling, as well as optimization strategies for scaling. We are also working on extending the implementation in that direction and intend to apply it to study corresponding examples involving resource management and load balancing. It is also interesting to investigate how the techniques developed here could be adapted to richer modelling languages for cloud-deployed software, such as ABS [21].

References

1. A. Abdelmaboud, D. N. Jawawi, I. Ghani, A. Elsafi, and B. Kitchenham. Quality of service approaches in cloud computing: A systematic mapping study. *Journal of Systems and Software*, 101:159 – 179, 2015.
2. O. Akar. Model Checking Of Ambient Calculus Specifications Against Ambient Logic Formulas. Bachelor's thesis, Istanbul Technical University, 2009.
3. B. Aman and G. Ciobanu. Mobile ambients with timers and types. In C. B. Jones, Z. Liu, and J. Woodcock, editors, *Proc. 4th Intl. Colloquium on Theoretical Aspects of Computing (ICTAC'07)*, *LNCS* 4711, pages 50–63. Springer, 2007.
4. B. Aman and G. Ciobanu. Timers and proximities for mobile ambients. In V. Diekert, M. V. Volkov, and A. Voronkov, editors, *Proc. 2nd Intl. Symp. on Computer Science in Russia (CSR'07)*, *LNCS* 4649, pages 33–43. Springer, 2007.
5. J. C. M. Baeten and J. A. Bergstra. Real Time Process Algebra. Technical Report CS-R 9053, Centrum voor Wiskunde en Informatica (CWI), 1990.
6. J. C. M. Baeten and C. A. Middelburg. *Process Algebra with Timing.* Monographs in Computer Science. An EATSC series. Springer, 2002.
7. M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B. Yassour. The Turtles project: Design and implementation of nested virtualization. In *Proc. 9th USENIX Symp. on Operating Systems Design and Implementation (OSDI 2010)*, pages 423–436. USENIX Association, 2010.
8. M. Berger. *Towards Abstractions for Distributed Systems.* PhD thesis, University of London, Imperial College, 2004.
9. L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. 27th Symp. on Principles of Programming Languages (POPL '00)*, pages 365–377, ACM, 2000.
10. L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
11. L. Cardelli and A. D. Gordon. Logical properties of name restriction. In *Proc. 5th Intl. Conf. on Typed Lambda Calculi and Applications (TLCA 2001)*, *LNCS* 2044, pages 46—-60. Springer, 2001.
12. L. Cardelli and A. D. Gordon. Equational properties of mobile ambients. *Mathematical Structures in Computer Science*, 13(3):371–408, 2003.
13. W. Charatonik, S. Dal-Zilio, A. D. Gordon, S. Mukhopadhyay, and J. Talbot. The complexity of model checking mobile ambients. In F. Honsell and M. Miculan, editors, *Proc. 4th Intl. Conf. on Foundations of Software Science and Computation Structures (FOSSACS 2001)*, *LNCS* 2030, pages 152–167. Springer, 2001.
14. W. Charatonik and J. Talbot. The decidability of model checking mobile ambients. In L. Fribourg, editor, *Proc. of 15th Intl. Workshop on Computer Science Logic (CSL 2001)*, *LNCS* 2142, pages 339–354. Springer, 2001.
15. G. Ciobanu. Interaction in time and space. *Electronic Notes in Theoretical Computer Science*, 203(3):5–18, 2008.
16. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude — A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, *LNCS* 4350. Springer, 2007.
17. S. Crago, K. Dunn, P. Eads, L. Hochstein, D. I. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. P. Walters. Heterogeneous cloud computing. In *2011 IEEE Intl. Conf. on Cluster Computing*, pages 378–385, Sept 2011.

18. Fibonacci. Greedy algorithm for Egyptian fractions. Available at `https://en.wikipedia.org/wiki/Greedy_algorithm_for_Egyptian_fractions`.
19. R. P. Goldberg. Survey of virtual machine research. *IEEE Computer*, 7(6):34–45, 1974.
20. A. D. Gordon. V for virtual. *Electronic Notes in Theoretical Computer Science*, 162:177–181, 2006.
21. E. B. Johnsen, R. Schlatte, and S. L. Tapia Tarifa. Integrating deployment architectures and resource consumption in timed object-oriented models. *Journal of Logic and Algebraic Methods in Programming*, 84(1):67–91, 2015.
22. E. B. Johnsen, M. Steffen, and J. B. Stumpf. A calculus of virtually timed ambients. In P. James and M. Roggenbach, editors, *Proc. 23rd Intl. Workshop on Algebraic Development Techniques (WADT 2016)*, *LNCS* 10644, pages 88–103. Springer, 2017.
23. E. B. Johnsen, M. Steffen, and J. B. Stumpf. Virtually timed ambients: A calculus of nested virtualization. *Journal of Logical and Algebraic Methods in Programming*, 94:109 – 127, 2018.
24. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
25. M. Merro and F. Zappa Nardelli. Behavioral theory for mobile ambients. *Journal of the ACM*, 52(6):961–1023, 2005.
26. J. Meseguer. Twenty years of rewriting logic. *Journal of Logic and Algebraic Programming*, 81(7-8):721–781, 2012.
27. J. Meseguer and G. Rosu. The rewriting logic semantics project. *Theoretical Computer Science*, 373(3):213–237, 2007.
28. R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proc. ICALP '92*, *LNCS* 623, pages 685–695. Springer, 1992.
29. X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.
30. P. C. Ölveczky. *Designing Reliable Distributed Systems – A Formal Methods Approach Based on Executable Modeling in Maude*. Undergraduate Topics in Computer Science. Springer, 2018.
31. J. Ouaknine and J. Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3, 2007.
32. J. Ouaknine and J. Worrell. Some recent results in metric temporal logic. In *Proc. 6th Intl. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS 2008)*, *LNCS* 5215, pages 1–13. Springer, 2008.
33. C. Prisacariu. Timed distributed pi-calculus. In *Modelling and Verifying of Parallel Processes (MOVEP06)*, pages 348–354, 2006.
34. F. Rosa-Velardo, C. Segura, and A. Verdejo. Typed mobile ambients in Maude. *Electronic Notes in Theoretical Computer Science*, 147(1):135 – 161, 2006. Proc. 6th Intl. Workshop on Rule-Based Programming.
35. D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
36. J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *Proc. 21th Intl. Conf. on Computer Aided Verification (CAV'09)*, *LNCS* 5643, pages 709–714. Springer, 2009.
37. Y. Sun. Toward a Model Checker for Ambient Logic Using the Process Analysis Toolkit. Master's thesis, Bishop's University, Sherbrooke, Quebec, Canada, 2015.
38. D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize once, run everywhere. In *Proc. 7th European Conf. on Computer Systems (EuroSys'12)*, pages 113–126. ACM, 2012.