An Analysis Tool for Models of Virtualized Systems

E. B. Johnsen M. Steffen J. B. Stumpf L. Tveito

Abstract

This paper gives an example-driven introduction to modelling and analyzing virtualized systems in, e.g., cloud computing, using virtually timed ambients, a process algebra developed to study timing aspects of resource management for (nested) virtual machines. The calculus supports nested virtualization and virtual machines compete with other processes for the resources of their host environment. Resource provisioning in virtually timed ambients extends the capabilities of mobile ambients to model the dynamic creation, migration, and destruction of virtual machines. Quality of service properties for virtually timed ambients can be formally expressed using modal contracts describing aspects of resource provisioning and verified using a model checker for virtually timed ambients, implemented in the rewriting system Maude.

1 Introduction

Cloud computing is a paradigm of distributed computing which allows users to store data and execute processes in a shared pool of data centers. A key factor in the success of cloud computing is *virtualization* [7, 11]. Virtualization technology represents the resources of an execution environment as a software layer, a so-called virtual machine. It allows to share existing resources, improves security by providing isolation of different users sharing the same resource, and enables dynamic assignment of resources according to consumer demand. The sharing of resources creates business drivers which make cloud computing an economically attractive model for deploying software [4]. *Nested virtualization* [8] is crucial to support cloud systems, as it enables virtual machines to migrate between different cloud providers [22]. It is also necessary to host virtual machines with operating systems which themselves support virtualization [3], such as Microsoft Windows 7 and Linux KVM.

Virtually timed ambients [13] is a calculus of explicit resource provisioning, based on the well-known calculus of mobile ambients [6]. It can be used to model nested virtualization in cloud systems, as virtually timed ambients formalize explicit resource management for virtual machines. The time model used to realize the 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 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

This paper was presented at the NIK-2018 conference; see http://www.nik.no/.

ambient, as well as in the virtually timed ambient which is left, and in the virtually timed ambient which is entered.

In cloud computing, a service-level agreement is a contract between a cloud provider and a client, specifying properties the system has to satisfy with respect to *quality of service*, such as mean time between failures, responsibility for various data rates or resource consumption. As virtually timed ambients can model nested virtualization in cloud systems and *modal logic* can be used to define properties of such systems, modal contracts for virtually timed ambients [14] formalize quality of service statements about cloud systems modeled in virtually timed ambients. A simulator and a *model checker* for virtually timed ambient have been implemented in the Maude system, as a tool to prove that a system satisfies a given proposition [14]. Maude was chosen as execution platform as it provides an intuitive way to model distributed systems at a high level of abstraction [18].

This paper shows by examples how models of virtualized systems can be constructed in virtually timed ambients and analyzed with the model checker. Several concrete examples are analyzed to verify quality of service statements. Together, the basic building blocks of these examples constitute a library for developing models of virtualized systems.

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

- exploration of the *virtually timed ambient calculus* and the corresponding modal logic as a model for virtualization in cloud computing;
- a *library of basic building blocks* for modeling cloud systems in virtually timed ambients;
- *examples and analysis* of cloud models, using the model checker tool for virtually timed ambients.

Paper overview. We introduce virtually timed ambients, their implementation in Maude and the corresponding modal logic in Section 2. Section 3 describes a library of building blocks for cloud models in virtually timed ambients. Section 4 presents different examples of cloud architecture and analyses them using modal logic. We discuss related work and conclude in Sections 5 and 6.

2 Virtually Timed Ambients

Virtually timed ambients extend mobile ambients with notions of virtual time and resource consumption, in order to model aspects of virtualization in cloud computing. We first recapitulate the main points of the calculus of mobile ambients before discussing the enhancements made by the calculus of virtually timed ambients.

Preliminaries on mobile ambients. The ambient calculus is a process algebra of locations and domains, originally developed by Cardelli and Gordon [6] for distributed systems such as the Internet. Mobile ambients are processes with a concept of location, arranged in a dynamically evolving hierarchy. An ambient represents the location or domain where a process is running, as illustrated by Fig. 1.

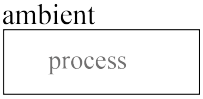

Figure 1: Graphical representation of a mobile ambient containing a process.

Ambients can be nested, such that a surrounding *parental ambient* contains *subambients*, and the nesting structure can change dynamically. This is specified by three basic capabilities. The input capability **in** *n* indicates the willingness of a process to move its parental ambient into an ambient named *n*, running in parallel with the parental ambient (illustrated by Fig. 2); the output capability **out** *n* enables an ambient to leave its surrounding ambient *n*; and lastly the capability **open** *n* allows to open an ambient named *n* which is on the same level as the capability. This syntax and the corresponding semantics are explained in detail in [6].

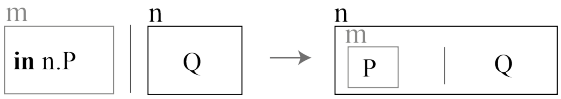

Figure 2: Graphical representation of the **in** *n* capability of mobile ambients.

Virtually timed ambients and their implementation. Mobile ambients are located processes, arranged in a hierarchy which may change dynamically. Interpreting these locations as a places of deployment, virtually timed ambients [12, 13] extend mobile ambients with notions of virtual time and resource consumption.

Timed processes differ from mobile ambients in that each virtually timed ambient contains, besides possibly further virtually timed subambients, a *local scheduler* (see Fig. 3). 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 round-robin way. This corresponds to a simple form of fair, *preemptive scheduling*, which makes the system's behavior sensitive to co-located virtually timed ambients and resource consuming processes. Technically, a local scheduler has a speed, relating externally received to internally emitted time slices; it contains counters to register the numbers of received and emitted time slices; and it contains sets of names of local ambients and processes which have been served a time slice by the scheduler in the current cycle, and those who have not, respectively.

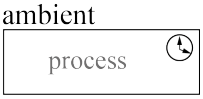

Figure 3: Graphical representation of a virtually timed ambient with a scheduler.

Timed capabilities extend the capabilities of mobile ambients by including a *resource consumption* capability, denoted **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, and the local schedulers need to be adjusted for every movement.

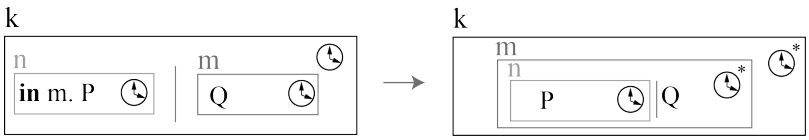

Figure 4: Graphical representation of the **in** *m* capability. The updated schedulers of the old and new parental ambient after the movement are marked with *.

Without adjusting the schedulers, the moving subambient would not receive time slices, which are represented with the notation `tick`, from the scheduler in its new surrounding ambient. For the **in** *n* and **out** *n* capabilities, the schedulers of the old and new surrounding ambient of the moving ambient are modified (as illustrated by Fig. 4). For the **open** *n* capability, the scheduler of the parent ambient itself is modified and the scheduler of the opened ambient is deleted. For the new consumption capability **c**, the time consuming process moves into the scheduler, where it waits to receive a time slice as resource before it can continue (see Fig. 5).

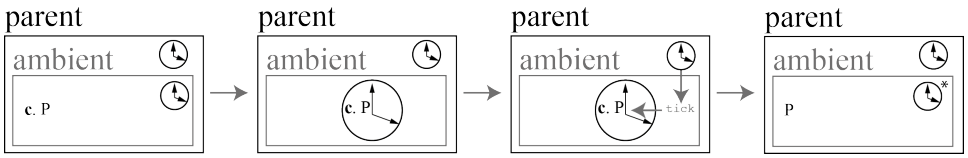

Figure 5: Graphical representation of the consume capability moving into the scheduler and consuming a resource after the ambient receives a `tick` from the parental ambient.

The calculus of virtually timed ambients has been implemented [14] in the Maude system for rewriting logic. Rewriting logic embeds *membership equational logic*, such that a specification or program can contain both equations and rewrite rules. When executing a Maude specification, rewrite steps are applied to normal forms in the equational logic. Both equations and rewrite rules may be *conditional*, meaning that specified conditions must hold for the rule or equation to apply. The Maude specification correlates directly to the formal definition of the calculus [13], hence we make use of the implementation to explain the calculus. The syntax of virtually timed ambients is represented by Maude terms, constructed from operators:[1]

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

Here all processes are defined with the data type `VTA`. The operator `zero` represents the inactive process, and parallel composition has the algebraic properties of being associative, commutative and having `zero` as identity element. Concatenation is represented with a dot, and virtually timed ambients are represented with a name followed by brackets, containing a scheduler and a process. Schedulers have a speed and contain the counters and sets used to control the distribution of time slices as outlined previously:

```
op sched_{_,_,_,_,_} : Rat Nat Nat Nat Servables Servables -> Scheduler.
```

[1]The full source code for the calculus and the examples described in this paper are available at: `https://github.com/larstvei/Check-VTA/tree/cloud-library`

The execution of timed capabilities is represented as *rewrite rules*, which are interpreted such that any term or subterm which matches the left hand side of the rewrite symbol => may be rewritten into the corresponding right hand side. Preconditions can be expressed using conditional rewrite rules, where a condition is stated after `if`. The **in** *m* capability, 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, Unserved, Served}
        | M[sched SpdM {InM, OutM, RestM, SrvM, NewParent} | R
          | N[sched SpdN {InN, OutN, RestN, SrvN, merge(UnSrvN, barb(P))}
            | P | Q]] | U]
      if  Unserved  := makeUnserved(N, UnSrvK, SrvK) /\
          Served    := makeServed(N, UnSrvK, SrvK) /\
          NewParent := makeParent(N, UnSrvK, SrvK, UnSrvM) .
```

Here the operations `makeUnserved`, `makeServed` and `makeParent` update the schedulers of the new and old parental ambients according to how the ambient was registered in the scheduler of the old parental ambient.

The execution of rewrite rules is represented in the syntax of the Maude tool by providing the rewriting command `rew` with a virtually timed ambient. The `rew` command applies the defined rewrite rules to the given ambient until termination, at which point the tool returns a `result`.

Example 1 (Virtually timed subambients and resource consumption) *A cloud server inside a system can be modelled by a virtually timed ambient* `'cloud`*, which here contains two* `tick` *and emits one time slice for every time slice it receives. It is entered by a virtually timed subambient* `'vm` *which needs to receive two time slices in order to enable resource consumption. This is simulated in the Maude tool as follows:*

```
rew 'system[sched 0 {0, 0, 0, none, 'vm}
      | 'cloud[sched 1 {0, 0, 0, none, none}
        | tick | tick]
      | 'vm[sched 1/2 {0, 0, 0, none, none}
        | in('cloud) . c . zero]] .

result VTA: 'system[sched 0{0,0,0,none,none}
              | 'cloud[sched 1{2,0,0,'vm,none}
                | 'vm[sched 1/2{2,0,0,none,none} | zero]]]
```

We can observe the movement of the virtual machine into the cloud, the consumption of the time slices as resources and the changing of the schedulers, which count the incoming time slices and gain (or loose, respectively) a subambient.

Modal logic for virtually timed ambients. Modal logic can be used to describe the behavior of systems. To capture resource provisioning in virtually timed ambients, we combine modal logic for mobile ambients [5] with notions based on metric temporal

logic [15, 19, 20] to define a modal logic for virtually timed ambients [14]. The validity of formulas is defined with regards to the calculus of virtually timed ambients by a satisfaction relation. In the Maude implementation, terms representing logical formulas are built from operator declarations, and the satisfaction relation becomes

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

The semantics of the satisfaction relations is expressed as a set of equations and rewrite rules. For example, a process `P` satisfies the *negation* of a formula `F` if and only if `P` does not satisfy `F`. This case is implemented by the following equation:

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

The *consumption* formula `Consume` is satisfied by any process `P` which contains a consumption capability. In the implementation, an operation `consumptions`, which is reduced by equations, determines if a process contains consume capabilities:

```
eq [Consumption] : P |= Consume = consumptions(P) .
```

The *sometime modality* `<> A @ N F` is satisfied by a process `P` if and only if `P` can reduce to a process satisfying the formula `F`, and uses less than `A` resources in the ambient named `N` during this reduction. 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 `distance` and `contains` define the number of used resources and the existence of the name in the given process, and are reduced by equations. 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.

The remaining formulas of the modal logic for virtually timed ambients are implemented similarly to the instances given above. The resulting Maude program can easily be used to check modal properties for virtually timed ambients and is demonstrated in the following example.

Example 2 (Implementation of modal contracts for virtually timed processes) *We consider a cloud server containing a virtual machine* `'vm`*, which is entered by an application* `'app`*, similar to Example 1. We check if the system satisfies a quality of service contract stating that the application can be executed after the use of two time slices.*

```
rew 'cloud[sched 1 {0, 0, 0, none, 'vm}
      | tick | tick
      | 'app[sched 0 {0, 0, 0, none, none}
        | in('vm) . c . zero]
      | 'vm[sched 1/2 {0, 0, 0, none, none}
        | open('app) . zero]]
    |= <> 2 @ 'cloud ~ Consume .

result Bool: true
```

The model checker confirms that there exists a reduction path where after the use of two time slices in the cloud ambient, there is no consume capability left. This means that at most two time slices are needed to execute the application in the virtual machine.

3 A Library for Cloud Models in Virtually Timed Ambients

In order to use virtually timed ambients as a modelling language for cloud computing, we develop a library containing important elements of cloud architecture, which can be composed according to a modular principle.

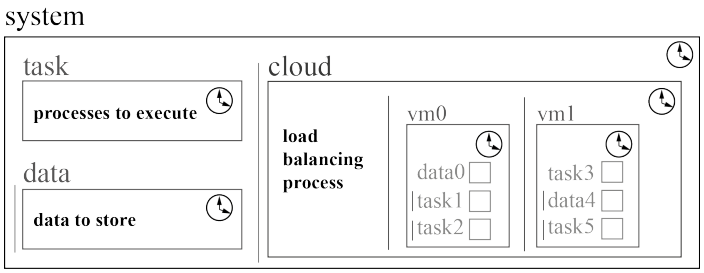

Figure 6: Example configuration of a cloud model.

A cloud model consists of a system containing a cloud and several tasks or data packages. The cloud typically includes a load balancing or scaling process, depending on the chosen load balancing strategy, as well as virtual machines. Tasks and data packages enter the cloud in order to be executed or stored. Here, let `sdl` represent an empty scheduler with no speed and let `(s K)` denote the sucessor of a natural number `K`.

System. The system is the outermost global level in which all computation takes place. Every process or resource used during the computation must be installed in this system.

```
'system[sdl | ... ]
```

Cloud. The `cloud` ambient models the cloud level, and contains the scaling or load-balancing process,the virtual machines and a number `K` of resources in the form of time slices. It is entered by tasks and data.

```
'cloud[sched 1 {0, 0, 0, none, none} | time-slices(K) | ... ]
```

Resources and consumption. Resources are given in the form of time slices `tick` and are exhausted by the consume capability.

```
eq time-slices(0) = zero .
eq time-slices(s K) = tick | time-slices(K) .

eq consumes(0) = 'done[sdl | zero] .
eq consumes(s K) = c . consumes(K) .
```

Tasks and data. Tasks enter the cloud and request processing resources in order to execute, while data expends memory capacity while it is stored and can be retrieved again.

```
eq task(K) =
    'task[sdl | in('cloud) . open('move) . zero | consumes(K)] .
```

Round-robin load balancing. In round-robin load balancing, a defined set of virtual machines receives incoming tasks and data in a round-robin way with the help of a load-balancer. The virtual machines can be defined with a name and a speed.

```
eq virtualMachineRR(X, Speed) =
    X[sched Speed {0, 0, 0, none, none}
      | ! (open('task) . open('done) . zero)] .
```

The cloud ambient needs to contain the following contents for round-robin load balancing, where `round-robin-lb(Ns)` describes the load balancing process and `createRRVMs(Ns, Rs)` creates the virtual machines:

```
eq round-robin(Ns, Rs) = round-robin-lb(Ns) | 'round_lock[sdl | zero]
              | createRRVMs(Ns, Rs) | 'load_balancer_lock[sdl | zero] .
```

Competing virtual machines. In this scenario the virtual machines report to the load-balancing process when they are empty and the load-balancer nondeterministically choses one of the idle machines to process the next task or data package.

```
eq virtualMachineI(X, Speed) =
    X[sched Speed {0, 0, 0, none, none}
      | idle(X)
      | ! (open('task) . open('done) . idle(X))] .
```

To deploy competing virtual machines the cloud ambient needs to contain a process `createIVMs(Ns, Rs)` to create the virtual machines and a suitable load balancer:

```
eq idling(Ns, Rs) = createIVMs(Ns, Rs) | load-balancer .
```

Auto-scaling. Here a `scaling` process creates a new virtually timed ambient with a restricted name and predefined speed for each task, similar to lightweight container systems [16]. After the task has been executed, the virtually timed ambient moves into the `garbage`.

```
eq virtualMachineAS(X, Speed) =
    X[sched Speed {0, 0, 0, none, none}
      | 'move[sdl |  out(X) . in('task) . in(X) .
            'scaling_lock[sdl | out(X) . zero]]
      | open('task) . zero
      | open('done) . in('garbage) . zero] .
```

For auto-scaling the cloud ambient needs to contain an ambient for garbage collection, the scaling process that creates the virtual machines `scaling(Speed)` and a lock:

```
eq auto-scaling(Speed) =
    garbage | scaling(Speed) | 'scaling_lock[sdl | zero] .
```

4 Analysis of Cloud Models in Virtually Timed Ambients

We present and analyze three examples inspired by cloud computing, which deploy load balancing and scaling, as introduced in Section 3, in different ways. In each case we simulate load on the machines and use a particular data package called `observable` to examine the satisfaction of certain quality of service statements by the model.

```
eq observable(K) =
    'task[sdl | open('move) . zero
      | 'observable[sched 1 {0, 0, 0, none, none}
        | consumes(K) | open('done) . zero]] .
```

The following modal logic formulas represent some essential quality of service statements in cloud computing scenarios. The first formula expresses that after the computation has been completed there are `K` idle virtual machines in the cloud:

```
eq F1(K) =
  <> 0 @ 'system 'system[~ Consume /\ (<> 0 K ('isCloud[True] | True))] .
```

As a higher number of virtual machines leads to higher energy consumption, this is necessary information for the calculation of energy costs. This formula does not only make use of the *sometime* modality `<> A @ N F` but also the *somewhere* modality `<> Speed K F`, describing relative change in speed and the number of subambients in the nested ambient satisfying the formula `F`. The next formula states that after using `K` resources, the execution of all tasks on the cloud will have completed.

```
eq F2(K) = <> K @ 'cloud (~ Consume) .
```

This is significant information regarding the required CPU and memory performance as well as resource allocation. The third formula is about the availability and response time of the system, and states that `K` resources are needed to execute the consume capabilities in the observable data package.

```
eq F3(K) =
  <> K @ 'cloud 'system['cloud[(+) ('observable[~ Consume] | True)]] .
```

By checking the satisfaction of these formulas, we can easily make quantitative statements about different cloud computing scenarios.

The following three examples model typical behavior in cloud systems, including migration, load balancing, scaling, locking and garbage collection. In order to provide minimal examples of load balancing we initialise the models with two virtual machines and add one task and the observable data package to simulate load on the machines. To focus on the core behavior of the example, we omit the movement of the tasks and data packages into the cloud and start directly with a configuration where they are already inside. The examples are given as follows:

```
eq example(P, Ns) =
       'system[sdl | 'cloud[sched 1 {0, 0, 0, none, Ns}
           | P | 'isCloud[sdl | zero]
           | time-slices(5)
           | createTasks(1, 1)  | observable(1)]] .
```

An instance of round-robin load balancing. We initialize the system with a fixed set of virtual machines, which receive the incoming tasks from the load balancer in order.

```
rew example(round-robin(('vm0 : 'vm1), (1 1)), ('vm0 : 'vm1)) |= F1(2) .
result Bool: true
rew example(round-robin(('vm0 : 'vm1), (1 1)), ('vm0 : 'vm1)) |= F2(2) .
result Bool: true
rew example(round-robin(('vm0 : 'vm1), (1 1)), ('vm0 : 'vm1)) |= F3(1) .
result Bool: true
```

By modifying the formulas we can check if the model can terminate with fewer running virtual machines or if it can run with fewer resources:

```
rew example(round-robin(('vm0:'vm1),(1 1)),('vm0:'vm1))|=F1(1)\/F2(1).
result Bool: false
```

Virtual machines compete for tasks. In this model we initialize the system with a fixed set of virtual machines competing for incoming tasks by communicating with the load-balancer, which assigns tasks nondeterministically.

```
rew example(idling(('vm0 : 'vm1), (1 1)), ('vm0 : 'vm1)) |= F1(2) .
result Bool: true
rew example(idling(('vm0 : 'vm1), (1 1)), ('vm0 : 'vm1)) |= F2(2) .
result Bool: true
rew example(idling(('vm0 : 'vm1), (1 1)), ('vm0 : 'vm1)) |= F3(1) .
result Bool: true
```

Auto-scaling on the cloud. This model is initialized without pre-defined virtual machines. Instead, the auto-scaling process creates a new virtual machine for each incoming task or data package.

```
rew example(auto-scaling(1), none) |= F1(1) . result Bool: true
rew example(auto-scaling(1), none) |= F2(2) . result Bool: true
rew example(auto-scaling(1), none) |= F3(1) . result Bool: true
```

In all three cases it holds that two resources are needed to execute the tasks on the machines, while one resource is needed to respond to the observable data package. This result is unsurprising as all virtual machines in the given models are initialized with the same speed. However, in the first two models there are two running virtual machines at the end of the computation, as the number of virtual machines is fixed in these scenarios, while in the third model the number of virtual machines has been reduced to one. This makes the auto-scaling model the most energy efficient of the given models.

5 Related Work

Virtually timed ambients were first defined to study bisimulation for virtual resource management [12]. They are based on mobile ambients [6], which model location mobility for processes executing in distributed, hierarchical networks. Gordon proposed a simple formalism for virtualization loosely based on mobile ambients [9]. Virtually timed ambients [12, 13] are closer to the syntax of the original mobile ambient calculus, while at the same time including notions of *time* and *explicit resource provisioning*.

Previous research on time in the ambient calculus [1] and in process calculi in general [2, 10, 17] focuses mostly on time-out behavior, i.e., the stalling of a process after a certain amount of time, by adding timers and a global clock to the calculus. We

take a complementary viewpoint by considering local schedulers, which allows different locations to have different speeds and focus on processing power and the question of how many tasks can be solved in a given amount of time by a system.

Modal logic for mobile ambients was introduced to describe properties of spatial configuration and mobile computation for a fragment of mobile ambients without replication and restriction on names [5]. We combine this logic with ideas from metric temporal logic [15, 19, 20] to specify notions of time and resources [14].

The operational reduction rules for mobile ambients as well as a type system have been implemented in Maude before [21]. In contrast, our implementation focuses on capturing the timed reduction rules of virtually timed ambients as well as modal formulas to enable model checking.

6 Concluding Remarks

Virtualization opens for new and interesting formal computational models. This paper presents virtually timed ambients in terms of a tool and a library of building blocks for modelling cloud systems. The tool is implemented in the Maude system for rewriting logic and can be used to develop and analyse models of virtualization in cloud computing.

The calculus of virtually timed ambients is 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. We provide a modal logic for the calculus and implement a model checker in the Maude rewriting logic system. By considering modal propositions as quality of service statements, we can model cloud systems and prove whether they satisfy certain service-level agreements expressed in modal logic.

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 this direction, and intend to apply it to study corresponding examples involving resource management and load balancing.

References

[1] B. Aman and G. Ciobanu. Mobile ambients with timers and types. In *Theoretical Aspects of Computing - ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, pages 50–63, 2007.

[2] J. C. M. Baeten and J. A. Bergstra. Real Time Process Algebra. Technical Report CS-R 9053, Centrum voor Wiskunde en Informatica (CWI), 1990.

[3] 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 *Proceedings 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2010)*, pages 423–436. USENIX Association, 2010.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[5] L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '00, pages 365–377, New York, NY, USA, 2000. ACM.

[6] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[7] T. S. Dillon, C. Wu, and E. Chang. Cloud computing: Issues and challenges. In *24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010)*, pages 27–33. IEEE Computer Society, 2010.

[8] R. P. Goldberg. Survey of virtual machine research. *IEEE Computer*, 7(6):34–45, 1974.

[9] A. D. Gordon. V for virtual. *Electronic Notes in Theoretical Computer Science*, 162:177–181, 2006.

[10] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.

[11] R. Jain and S. Paul. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, 51(11):24–31, 2013.

[12] E. B. Johnsen, M. Steffen, and J. B. Stumpf. A calculus of virtually timed ambients. In P. James and M. Roggenbach, editors, *Postproceedings of selected contributions to the 23rd International Workshop on Algebraic Development Techniques (WADT 2016)*, volume 10644 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2017.

[13] 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.

[14] E. B. Johnsen, M. Steffen, J. B. Stumpf, and L. Tveito. Checking modal contracts for virtually timed ambients, 2018. Submitted for publication. Available at `www.ifi.uio.no/~johanbst/MLCheckingVTAS.pdf`.

[15] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[16] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), Mar. 2014.

[17] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J. C. M. Baeten and J. W. Klop, editors, *Proceedings 1st International Conference on Concurrency Theory (CONCUR'90)*, volume 458 of *Lecture Notes in Computer Science*, pages 401–415. Springer, 1990.

[18] P. C. Ölveczky. *Designing Reliable Distributed Systems – A Formal Methods Approach Based on Executable Modeling in Maude*. Undergraduate Topics in Computer Science. Springer, 2018.

[19] J. Ouaknine and J. Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3, 2007.

[20] J. Ouaknine and J. Worrell. Some recent results in metric temporal logic. In F. Cassez and C. Jard, editors, *Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2008)*, volume 5215 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2008.

[21] F. Rosa-Velardo, C. Segura, and A. Verdejo. Typed mobile ambients in Maude. *Electronic Notes in Theoretical Computer Science*, 147(1):135 – 161, 2006. Proceedings of the 6th International Workshop on Rule-Based Programming.

[22] D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize once, run everywhere. In *Proceedings 7th European Conference on Computer Systems (EuroSys'12)*, pages 113–126. ACM, 2012.