Virtually Timed Ambients

A Calculus for Resource Management in Cloud Computing

Doctoral Dissertation by

Johanna Beate Stumpf

Submitted to the
Faculty of Mathematics and Natural Sciences at the University of Oslo
for the degree Philosophiae Doctor in Computer Science

Date of submission: 15. 06. 2018
Date of public defense: 11. 10. 2018

Analytical Solutions and Reasoning
Department of Informatics
University of Oslo
Norway

Oslo, June 2018

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 2022

ISSN 1501-7710

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

Abstract

Cloud computing is a paradigm of distributed computing in which users share resources by storing data and executing processes in common data centers. A key factor for the success of this paradigm is virtualization technology, which represents the resources of an execution environment as a software layer, a so-called virtual machine. Virtualization allows to share existing hardware and software resources, improves security by providing isolation of different users, which share the same resource, and enables dynamic assignment of resources according to the demand of the user. The sharing of resources creates business drivers which make cloud computing an economically attractive model for deploying software.

This thesis introduces the calculus of virtually timed ambients, a formal model of hierarchical locations for execution with explicit resource provisioning. This calculus is based on the well-known calculus of mobile ambients and motivated by the use of nested virtualization in cloud computing applications. The investigation of cloud computing from the point of view of process calculi provides a formal specification of the subject, which is necessary in order to develop executable models for analysis and optimization.

The main contributions of this thesis are the definition of the calculus of virtually timed ambients, and the reasoning about its essential characteristics. In order to enable static analysis we enhance the calculus with a type system. Furthermore, we define a modal logic and a corresponding model checker, which we deploy in the definition of resource-awareness of virtually timed ambients, enabling dynamic self management of processes. Lastly, we present virtually timed ambients as a framework to analyse virtualization in cloud computing utilizing a prototype implementation. All concepts are illustrated by examples.

Acknowledgments

First and foremost, I would like to thank my supervisors Einar Broch Johnsen and Martin Steffen for their continuous support and encouragement over the past years. They provided the right balance between guidance and independence for my research, while always taking time for my questions. I am deeply grateful to them for our many joint discussions, where they shared their deep insights, constructive suggestions and helpful advice.

I would like to extend my gratitude to all colleagues in the *Analytical Solutions and Reasoning* group for their support of my growth as a researcher and for always creating a welcoming, friendly and cooperative working environment. In particular, I am grateful to Lars Tveito for the collaboration on the implementation of the calculus.

Furthermore, I would like to thank the initiators and participants of the SIRIUS project for establishing fantastic opportunities to meet people and broaden my horizon, especially through the mentoring program.

My sincere thanks go to Sigurd Kittelsen, Yvonne Späck-Leigsnering, Lars Tveito and Felix Wolf for reading and commenting on the first draft of this thesis.

Lastly, I would like to thank my friends and family, for their encouragement and support during my studies. Danke.

The work presented in this thesis was funded by the Research Council of Norway through the CUMULUS project on semantics-based analyses for cloud-aware computing.

Contents

Part I

Overview

CHAPTER 1

Motivation

Cloud computing is a paradigm of information technology that describes the shared use of resources, applications, and services over the Internet. A key factor for the success of cloud computing is virtualization. Virtualization technology represents the resources of an execution environment as a virtual machine. This allows to share resources, improves security by providing isolation of different users and enables dynamic pay-on-demand assignment of resources. The sharing of resources creates business drivers which make cloud computing an economically attractive model for deployment of software and data storage. In 2015 the EU estimated that cloud based data processing will create 2.5 million new jobs and an annual value of 160 billion euro in Europe by 2020 [57].

The reduction of costs due to resource allocation on demand is only one of the benefits claimed by cloud providers. Cloud computing further improves the agility and productivity of organizations, as multiple users can work on the same data simultaneously. Maintenance gets easier as device-independent online applications allow users to access data and systems via their web browser, regardless of their location or equipment. Reliability can be improved with the use of redundancy and security can be improved through the centralization of data.

However, the necessity to hand the control of data and resources over to service providers, in order to reduce the complexity of data management and the costs for the operation of servers, creates conflicts with the need to keep control over sensitive data and the need to ensure optimal computing performance. To overcome these conflicts and to regain control of the virtualized resources on the cloud, a full formal specification of the subject is necessary.

In this thesis, we aim to develop a formal foundation for virtual environments in order to enable the formal analysis of virtualization in cloud computing through the means of formal methods. The result of our research is the *calculus of virtually timed ambients* presented in this thesis.

1.1 Introduction to Cloud Computing

In the following we present a short history of cloud computing before introducing the fundamental properties which characterize the paradigm, as well as the most commonly applied service and deployment models. Furthermore, we discuss the role of virtualization in cloud computing.

Historical Context

The principle of cloud computing has become widespread only in the last decade but the idea of sharing resources goes back almost as far as the digital computer itself. In the 1950s, when industry and governments started to make use of computers, a single mainframe was highly expensive and big enough to fill a room. Thus, users shared the resource of the single computer by sharing time and data access. An automated form of time-sharing was introduced in the 1960s [17,124] by connecting computer terminals to an institutional mainframe. This gave users the illusion of working on a stand-alone computer, while actually sharing resources by means of multiprogramming and multitasking. However, as processors became smaller and cheaper, users were able to afford their own computer and this form of resource sharing fell into oblivion.

The use of personal computers became more popular during the 1980s, and while at that time it was no longer necessary to share physical machines, the idea of time-sharing evolved further into the concept of virtual machines [38,72], allowing multiple separate virtual environments to exist at the same time on the same physical machine. In the 1990s the Internet opened up a whole new digital world to the general public. This development accelerated in the next years and the production and processing of larger and larger masses of data, and applications brought the concept of resource sharing back to the forefront of information technology [130,138]. This time, resources and services were made available through the Internet and the name 'cloud computing' was coined.

During the last two decades, different providers have offered a variety of on-demand online services utilizing the sharing of virtual resources. Cloud services by Amazon and Microsoft are well-known in an industry context, whereas the examples of cloud computing probably best known to the general public are Dropbox and Google Docs. Lately, the launch of cloud services by several large technology companies in combi-

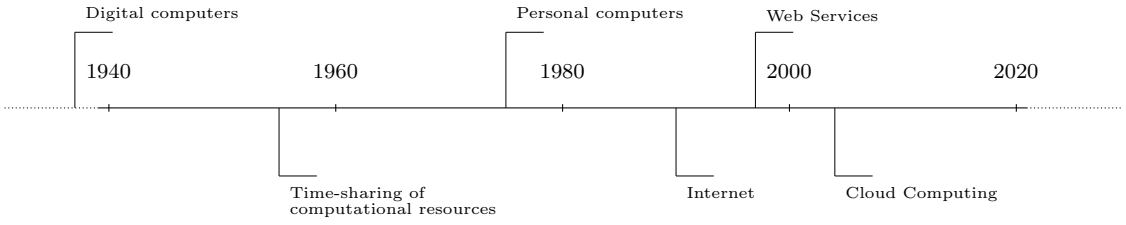

Figure 1.1: Timeline of digital computing.

nation with the availability of high-capacity networks and low-cost computers, led to cloud computing becoming the default information infrastructure in industry [37,105].

While the idea of sharing resources has been around since the 1950s and cloud services have been successfully monetized in the last years, the topic of cloud computing remains a subject of active research. In the following we give a characterisation of the main properties of cloud computing and relate them to the research presented throughout this thesis.

Characteristics

The National Institute of Standards and Technology of the U.S. Department of Commerce identifies five essential characteristics of cloud computing [107]:

> "1. *On-demand self-service.* A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
> 2. *Broad network access.* Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
> 3. *Resource pooling.* The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.
> 4. *Rapid elasticity.* Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
> 5. *Measured service.* Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service."

In our work we will presuppose the existence of network access. All other general characteristics of cloud computing can be modeled by the calculus of virtually

timed ambients. Even though these characteristics lead to a very broad description of cloud computing, there exist established standards allowing more precise definitions. To specify a certain cloud computing scenario one usually differentiates between the following common models of service and deployment scenarios.

Service Models. Cloud computing providers offer different models of services to their customers. The main service models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [105, 107], which are visualized in Figure 1.2. The services delivered by a provider to a costumer are regulated in contracts called *service-level agreements* [158], specifying different properties that need to be fulfilled by the service provider, for example with respect to quality of service, privacy, response time or availability. The work in this thesis focuses on agreements regarding quality of service.

Figure 1.2: Infrastructure, platform and software as a service.

Infrastructure as a Service. The provider offers infrastructure including servers, virtual machines, storage, network and operating systems. Customers can access the virtual machines to deploy and run arbitrary software as desired, as it is for example possible with Amazon Cloud Services or Microsoft Azure.

Platform as a Service. Providers host software development tools on their infrastructure, which can be used by customers to develop new software on-demand. The newly created applications are managed and deployed on the same cloud infrastructure, as for example in Google's Firebase, or in IBM Bluemix.

Software as a Service. Ready-made software is offered by the provider over the Internet. Costumers can access these web services on-demand from anywhere using any device with suitable Internet access. In this case the online service Dropbox or software offered by SAP are well-known examples.

In principle, our work on the calculus of virtually timed ambients supports the modelling of all given service models. A library of basic building blocks for cloud computing, which is discussed towards the end of this thesis, can be considered an abstraction of a platform, as it allows the user to build new software for the cloud.

Deployment Models. Deployment models are used to define location and access rights of cloud servers. While this topic of security is not directly addressed in this thesis, it can be considered for future work on the calculus. The main deployment models for cloud computing are public cloud, private cloud and hybrid cloud [105,107], and can be seen in Figure 1.3.

Figure 1.3: Public, private and hybrid cloud.

Public cloud. A third-party cloud service provider delivers cloud services and resources over the Internet. The costumers pay on-demand for storage or bandwidth. This model offers low costs and high flexibility.

Private cloud. A private cloud is hosted on a company's own servers and services are only delivered to internal users. This model provides the benefits of the cloud, while strongly emphasizing control over data and security.

Hybrid cloud. A hybrid cloud is a combined use of private and public cloud services in the sense that sensitive information is stored privately, while high workload may be processed on public cloud servers. This model requires appropriate regulations to allow data and applications to move between private and public cloud while protecting critical information.

Virtualization

Virtualization describes the creation of a virtual version of technological resources like servers or operating systems. It is the main enabling technology for cloud computing as the representation of resources through virtual machines allows sharing of existing resources, improvement of security by isolating different users sharing the same resource and enabling dynamic assignment of resources according to consumer demand [60, 83]. This way virtualization creates an easily scalable system of independent virtual machines. Furthermore, virtualization increases the agility of a system as it allows to reallocate virtual resources quickly without the handling of actual physical resources, for example servers. Figure 1.4 visualizes the restriction of resources on the hardware

Figure 1.4: Representation of resources on hardware, software and virtualization level.

level, which are fixed, and on the software level, which are not dynamic, as opposed to virtualized resources, which can change dynamically according to demand. As idle virtual machines can be allocated and deployed where they are needed, virtual resources can be used more efficiently than physical ones. The effective sharing of virtual resources with pay-on-demand resource provisioning rather than the upfront investment in servers creates the business drivers which make cloud computing an economically attractive model for deploying software [37].

Software processes are agnostic to whether they run on a virtual machine or directly on physical hardware. A virtual machine is itself such a process, which can be executed on another virtual machine. This form of nested virtualization [73] is a crucial technology to support the heterogeneous cloud [58], as it enables virtual machines to migrate between different cloud providers [159]. It is also necessary to host virtual machines with operating systems which themselves support virtualization [19], such as Microsoft Windows 7 and Linux KVM. The importance of nested virtualization is one of the basic ideas behind our work on virtually timed ambients. Furthermore, as virtualization enables the more effective use of existing resources, it can help to reduce the total power usage of data processing, This way, it provides a favorable environmental impact through reduced energy consumption.

1.2 Research Goal and Methodology

Virtual applications on the cloud can in principle modify resources of their own deployment scenario during execution, for example, to dynamically create virtual machines or reallocate resources to an existing virtual machine. This so-called *elasticity* of cloud computing turns effective deployment of virtual machines into a multi-objective problem. It is important to *maximize performance* by enhancing response time, simultaneously, the accumulated price for the leased virtual machines has to be taken into account, in order to *minimize cost*. To capture this concern, a model of cloud computing needs to account for pay-on-demand resource usage and to evaluate and compare different deployment scenarios with respect to these objectives. However, today there exist no general, systematic means beyond simulation to model and verify software in the context of virtualized resources, nor to analyze resource management for programmable infrastructure.

General-purpose modeling languages strive for abstraction in order to reduce complexity [97]. Descriptions primarily focus on the functional behavior and logical composition of software and ignore how the software's deployment influences its behavior. For embedded and cyber-physical systems, it is nowadays accepted that modeling and programming languages need timed semantics [99]. The inherent resource constraints of these systems have led to a large body of work extending formal models with time, including process algebra, Petri nets, automata, and games. In the last years, similar extensions have been developed for restricted resources beyond time, such as priced timed automata [26]. Related approaches have been applied to web services and business processes with resource constraints (e.g., [65]). These approaches

typically abstract from data flow and rely on domain experts to declare the time or other resource cost associated with transitions. However, the price of execution on the cloud depends not only on the cost of transitions but also on the capacity of the deployment context, which is not easily captured in these approaches.

A formal modeling language which supports resource management on the cloud has recently been developed [7], but analysis is currently restricted to simulation. Simulation has the drawback of only analyzing some possible behaviors of a system, while model checking in this scenario suffers from well-known combinatorial explosion problems for systems of reasonable size. Thus, none of the cited works directly address the challenges raised by the verification of virtualized cloud applications. In particular, they do not model quantitative resources as data inside the system itself, which is a particular property of virtualized resources.

Research Goal. In order to develop executable models of virtualized systems, which can be used for analysis and optimization, a complete formal specification of the subject is necessary. As currently there exist no foundational model of the basic features of cloud computing such as scalability and resource management, we define the main goal of this thesis as follows:

> *We aim to develop a formalization of virtual environments to enable the formal analysis of virtualization in cloud computing.*

We will address this goal via the following four research questions:

1. *How can we formalize resource management between distributed locations and allow comparison of different systems?*
2. *How can we statically predict the behavior of a system to avoid runtime errors?*
3. *How can we define specifications for resource management as given, e.g., in service-level agreements, which allow automatic verification?*
4. *How can we enable systems in the defined formalism to react dynamically to changes during runtime and, consequently, to perform dynamic self-management?*

Methodology. To accomplish the formalization goal of this thesis we make use of *formal methods* to achieve a complete formal specification and analysis of the subject. Formal methods are mathematical techniques for this kind of specification, as well as for development and verification of software. The use of formal methods for software design contributes particularly to the reliability and robustness of a design. The formal methods chosen to investigate and analyze the four research questions are described below and visualized in Figure 1.5.

The first of those formal methods is the use of a formal system. A formal system is a mathematical model entailed from a logical foundation. Each formal system has

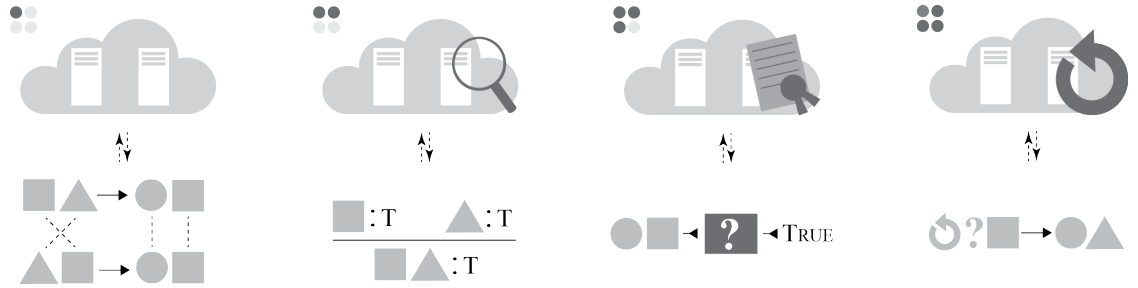

Figure 1.5: Visualization of the four research questions regarding formalization, statical analysis, service-level agreements and dynamic changes. The formal methods chosen to investigate and analyze those topics, namely process calculi, type systems, modal logic and resource-awareness, are illustrated underneath.

a formal language, which comprises the syntax of the language, consisting of primitive symbols, and the semantics, which are rules regulating the behavior of the symbols and their interactions with each other. The formal system then consists of any number of combinations of the symbols which are formed according to the semantics.

One such formal system is a *process calculus*. Process calculi constitute a tool for the high-level description of interactions, communications, and synchronizations between independent processes. They are minimal systems well-suited to study fundamental aspects of interacting processes. Process calculi further provide laws that allow process descriptions to be manipulated and analyzed, and permit formal reasoning about equivalences between processes. To formalize nested virtualization in cloud computing, notions of location mobility and nesting are essential. Thus, we aim to develop a calculus of virtualization, which does not only incorporate suitable notions of nested locations and mobility, but also local time and resources. The well-established calculus of mobile ambients focuses on processes executing at distributed locations and captures hierarchical nesting as well as mobility of explicit locations. This makes it an appropriate choice as a starting point for the modeling of cloud computing. We add a new notion of time to the mobile ambient calculus, to make it suitable for modeling cloud applications and enable resource distributing, thus defining the *calculus of virtually timed ambients*.

In order to enable analysis of the calculus, we define an equivalence relation in form of a *bisimulation*. In combination with the definition of the calculus this provides a suitable response to the first research question. Bisimulation relations over this kind of calculus allow comparisons of different systems via their behavior. Two systems are bisimilar if they can simulate each other's behavior. This way a bisimulation relation allows to define a system to be better than another one, for example in the sense of being similar but cheaper. Static analyses, as implied by the second research question, are compile time techniques used to avoid runtime errors. To treat this topic, we further enhance the calculus with a *type system* to enable static analysis. Type systems are a common technique to describe features of a calculus and provide a way to have

the implementation of those features statically checked. Moreover, in response to the third research question, we define a specification formalism for resource management, via *modal logic*, to allow the formal description of quality of service statements and service-level agreements. Additionally, we implement a model checker to have those properties automatically checked. Based on this modal logic we can approach the topic of the fourth research question by equipping the calculus with *resource-awareness* in order to enable dynamic self-management of processes during runtime.

1.3 Structure of this Thesis

This work is written in the form of a cumulative thesis, compiling a number of research papers. The thesis consists of two parts. Part I provides the necessary background and context for the research papers which are presented in full in Part II.

In the following, we introduce process calculi in general in Chapter 2 and point out different methods of reasoning about the behavior of processes. Chapter 3 focuses on the calculus of mobile ambients in particular as well as a bisimulation relation to compare different ambient systems. Summaries of all research papers and a list of additional publications are presented in Chapter 4. Chapter 5 contains a description of the contributions, re-visits the research questions and concludes with an outlook on future possibilities.

CHAPTER 2

Preliminaries on Process Calculi

The primary goal of this thesis is the development of a formal foundation for virtualized environments. To accomplish this goal and to enable the use of this foundation for the analysis of executable models of cloud systems, we utilize the formal method of process calculi. Process calculi (or process algebras) are formal languages with well defined semantics that permit describing and verifying properties of concurrent systems. The first person to use the phrase "algebra of processes" was Hans Bekič [18]. This was quickly followed by the invention of the calculus of communicating systems, CCS [113], and communicating sequential processes, CSP [79]. For a thorough introduction to the topic we direct the reader to [5, 16, 22].

2.1 Essential Features of Process Calculi

Process calculi get their name from the mathematical approach that is used to define processes. They were first introduced in the early seventies of the twentieth century to describe the semantics of programs containing concurrency [13].

Concurrent programs contain parts that can be executed in arbitrary order, without affecting the result of the computation. This allows for parallel execution of the concurrent parts. Parallel execution of processes makes programs highly complex, as several actions can happen at the same time. As minimal models of such complex systems, process calculi consist of a set of basic processes with interactions between them, including a parallel operator to represent concurrency. In general process calculi are also equipped with a transition system, describing the structural operational semantics, and a notion of equivalence.

Complex systems can be built from these basic building blocks by taking a compositional approach and defining the meaning of a more complicated system through the meaning of its smaller parts. This allows for an easy analysis of processes and enables formal reasoning about equivalences between processes, for example via bisimulation. Verification of concurrent systems can also be carried out through the addition of types for static analysis or via the checking of properties described by logical formulas.

To illustrate process calculi and different methods of reasoning, we introduce an artificial example calculus.

Example 1 (IN-calculus)**.** *In order to explain the features of process calculi we define a small example calculus with movement and nesting, which we call the* IN*-calculus. We define the syntax of processes in this calculus as follows:*

$$\begin{array}{rrll} P, Q ::= & \mathbf{0} & \text{termination} \\ \mid & P \mid Q & \text{parallel composition} \\ \mid & \text{IN}.P & \text{movement} \\ \mid & [P] & \text{target.} \end{array}$$

Here the process $\mathbf{0}$ *does nothing, defining the* termination *of a process.* Parallel composition $P \mid Q$ *describes concurrently running processes. Our calculus allows* movement *with the* $\text{IN}.P$ *activity, and the brackets* $[P]$ *describe the* target *of a movement activity. The semantics is given by reduction rules and equations defining structural congruence. Here the equations are defined as* $P \mid Q \equiv Q \mid P$ *and* $P \mid \mathbf{0} \equiv P$*, and the reduction rules of the* IN*-calculus are given as:*

$$\text{IN}.P \mid [Q] \rightarrowtriangle [P \mid Q], \qquad\qquad P \rightarrowtriangle P' \Rightarrow P \mid Q \rightarrowtriangle P' \mid Q.$$

This reduction rules allow processes to move non-deterministically into a target, which has to be located in parallel to the process. Thus, the movements of processes can create a nesting structure. This can be observed in the following example:

$$\text{IN}.\text{IN}.[\mathbf{0}] \mid [[\mathbf{0}]] \rightarrowtriangle [\text{IN}.[\mathbf{0}] \mid [\mathbf{0}]] \rightarrowtriangle [[[\mathbf{0}]]].$$

The definition of our calculus for resource management in cloud computing, which is introduced in this thesis, is based on the well-known calculus of mobile ambients [43]. The ambient calculus is defined similarly to the small example calculus given in this chapter, but contains further movement capabilities and named locations. Our calculus enhances the ambient calculus with notions of time and resources. In the following we discuss features of process calculi relevant to those properties before introducing different methods of reasoning.

Timed Process Algebras. Algebraic concurrency theories such as CCS, CSP and ACP [16, 79, 113] have been extended to deal with time-dependent behavior in various ways. Timing can either be absolute or relative and the time scale on which time is measured can be continuous, usually called real time, or discrete. Execution of actions and passage of time can either be separated or combined. One approach is to consider time to be real valued and semantically associate time directly with actions for instance via the use of timers. The other idea, which our approach shares with many others, is to introduce special actions to represent the passage of time. Separating actions from the passage of time corresponds to the two-phase scheme of modeling time-dependent behavior and combining action with the passage of time corresponds

to the time-stamping scheme. Timed process algebras which originated from ACP and CSP can be found in, e.g., [14,15,122]. As our work is based on the calculus of mobile ambients, we focus on closely related calculi like the π-calculus [146]. This calculus is connected to mobile ambients as they are both process calculi which emphasize names and mobility. The π-calculus originated from CCS. An early timed extension of CCS introduced a special action for time, without committing to a discrete or continuous time domain [118]. A related idling action σ is proposed in [77] such that processes in a standard process algebra would need exactly one time unit to process a σ, where time is discrete and processes synchronized via a global clock. A notion of local time for CCS is proposed in [147]; this notion resembles our model of local schedulers, but was realized in terms of a timeout oriented model. The authors extend their work by defining a real-time calculus for expressing delay in asynchronous communication [148]. The high-level idea in these works differs from the notion of time we introduce with the calculus of virtually timed ambients. All these approaches focus on speed as the duration of processes, while in our approach with local schedulers describes the processing power of a virtually timed ambient.

Timers have been studied for both the distributed π-calculus [20,137] and for mobile ambients [8,9,53]. Real-time extensions of CSP [140], ACP [16] and the π-calculus [101] follow a similar approach. In this line of work, timers which are controlled by a global clock are introduced to achieve the option of time-out. A similar path is followed in membrane computing [131], where the execution of each rule takes exactly one time unit, which is defined by a global clock. As membrane computing aims to model actual chemical reactions this is not sufficient, thus in timed P systems [48] each rule has an associated integer representing the time needed to complete execution of the rule. This resembles the timer approach on mobile ambients. In contrast, the schedulers in our work recursively trigger local schedulers in subambients which define the execution power of the nested virtually timed ambients. Modeling timeouts is a straightforward extension of our work.

Process Calculi and Mobility. An introduction to mobile process calculi can be found in [121]. We mention only the versions most relevant to our work. The basis for many mobility calculi is the π-calculus [146]. This calculus is closely related to the ambient calculus and therefore to our calculus of virtually timed ambients, as they both focus on names and mobility. However the notion of locations is very different. In the π-calculus the location of a process is defined by its network connections, while in the ambient calculus the locations are explicit. The Seal calculus [47] is based on the mobile ambient calculus and the two calculi are closely related [85]. However, in the Seal calculus interactions require explicit agreement between two agents, while in mobile ambients one ambient is active during an action, while the other ambient is passive. The Join calculus [66] can be seen as an asynchronous π-calculus with strong restrictions regarding scopes and reception. Based on this is the M-calculus [149], which can be interpreted as a version of the distributed Join calculus, and uses higher order communication. The Kell calculus [23] retains the advantages of the Seal calculus and the M-calculus, while preserving the simplicity of the ambient calculus. Calculi,

which are simpler to implement than the ambient calculus are sought-after in [23] and [52]. The δ-calculus in [52] uses synchronous movements in order to model distributed mobile real-time business applications. However, these calculi fail to preserve the simplicity of the ambient calculus.

Process Algebras with Resources. A calculus of Mobile Resources [71], which is loosely based on Boxed Mobile Ambients, considers processes inside slots, which resemble ambients, as resources. SCRP [139], which is based on CCS, models resources explicitly as primitives, similar to our approach. However, none of these calculi contain notions of time. ACSR [30] and its probabilistic extension PACSR [100] are algebras for the specification of distributed systems with resources and real-time constraints, and similar to our approach they distinguish between actions that take time and instantaneous processes. However, the timed aspects of the calculus focus on the implementation of time-outs and resources are elements of a finite set of reusable objects, analogous to locks. A similar approach is studied in [119], where the problem of possible deadlocks is solved by adding an explicit scheduling concept. In contrast, in our approach resources are distributed over time and scheduling is fixed in the implementation of the resource distribution. Resource-awareness for virtually timed ambients, enabling the dynamic reaction to changes during the reduction, is based on the Calculus of Context Aware Ambients [150] which introduces context-guarded processes to enable context-awareness of mobile ambients. We enhance the given context expressions to cover the timing and resource aspects of virtually timed ambients.

2.2 Reasoning about Process Calculi

The compositional semantics of process calculi allow to define the meaning of large systems in terms of the definition of their components. This permits complex processes to be analyzed via basic operators and rules. In the following we describe common methods to reason about the behavior of a system in terms of equivalence, type analysis, and dynamic satisfaction of properties.

Behavioral Equivalence

The behavior of a system consists of its data and processes. Two systems behave in the same way if they cannot be distinguished from each other. The criteria for this distinction can be defined in various ways, according to the desired differentiation.

Bisimulation equivalence [114, 129] defines two systems as equivalent if they can not only perform the same sequence of actions, but also, after each action has been executed, they again exhibit the same possible behavior. Thus, bisimulation defines an *equivalence relation*, which is reflexive, symmetric and transitive. In order to define bisimulation, one must first define the behavior of a system as labelled transitions, built on structural operational semantics, which are composed of the processes and the operators in the process calculus. Then a bisimulation can be defined as a binary

relation between two state transition systems, associating systems that behave in the same way in the sense that each of the systems can simulate the other. However, not all systems that can simulate each other are necessary bisimilar, as it additionally has to hold that the simulations back and forth are able to relate the same sets of states.

Weak bisimulation is defined by specifying actions, which are not observed by the relation. By using *up-to proof techniques* the definition of a bisimulation is relaxed in order to relate processes which are bisimilar up to a certain requirement. The definition of bisimulation can also be made stronger by making use of *barbs*, which are observables of terms. If the bisimulation relation defines an equivalence relation that is compatible with the structure of the underlying algebra, it is called a *congruence*.

In a game theoretical setting, bisimulation can be considered as a game between an attacker and a defender. Here, two systems are bisimilar if the two players can always match each other's moves. The attacker starts by choosing any valid transition in the system. Afterwards the defender attempts to match that transition. The systems are bisimilar if and only if there exists a winning strategy for the defender, which means that the game does not terminate.

Example 2 (In-calculus: Bisimulation)**.** *We define bisimulation for the* In*-calculus, by describing two processes as bismilar if they can simulate each others movements. In order to do this we define the labeled transition* $\xrightarrow{\text{IN}}$ *as* $\text{IN}.P \mid [Q] \xrightarrow{\text{IN}} [P \mid Q]$.

Now, we can define a symmetric relation $\mathcal{R}$ *over processes in the* In*-calculus as a* bisimulation *if* $P \mathcal{R} Q$ *implies that if* $P \xrightarrow{\text{IN}} P'$ *then there exists* Q' *such that* $Q \xrightarrow{\text{IN}} Q'$ *and* $P' \mathcal{R} Q'$*. Two processes* P *and* Q *are* bisimilar*, written* $P \approx_{\text{IN}} Q$*, if* $P \mathcal{R} Q$ *for some bisimulation* $\mathcal{R}$*.*

As there exists only one label due to the simplicity of the In*-calculus, the bisimulation relation in this case is not able to provide insights in different branching structures of the reduction.*

We reconsider the example process $P = \text{IN}.\text{IN}.[\mathbf{0}] \mid [[\mathbf{0}]]$ *from Example 1 and define a second process* $Q = \text{IN}.[\mathbf{0}] \mid [\text{IN}.[\mathbf{0}]]$*. This process reduces as follows:*

$$\text{IN}.[\mathbf{0}] \mid [\text{IN}.[\mathbf{0}]] \twoheadrightarrow [[\mathbf{0}] \mid \text{IN}.[\mathbf{0}]] \twoheadrightarrow [[[\mathbf{0}]]].$$

The reduction of P *can be seen in Example 1. Following the definition above it holds that the processes* P *and* Q *are bisimilar* $P \approx_{\text{IN}} Q$*, as* P *can match every* IN *step of* Q *and vice versa.*

Type Systems

A type system [46, 136] associates the values and processes in a system with a type. The type is a special property that can formalize the flow of information in a system as well as special features of processes. It enables the statical and safe approximation of the set of behaviors that can arise when a system is running. Thus, the use of a static type system enables the finding of possible runtime errors already during compile time.

The properties expressed by types are captured by context sensitive judgments, which are associated with the processes via a set of interference rules, which in turn

depend on the process calculi's syntax and semantics. The reduction relation of the calculus and the type system have to cohere in such a way that a well typed system is unable to produce an error during reduction, which includes that the reduction must be able to progress.

Example 3 (IN-calculus: Type system)**.** *We define a simple type system for the* IN*-calculus, which counts the number of nested locations. Process types in this case are natural numbers $n \in \mathbb{N}_0$. We define the typing rules as follows:*

$$\frac{}{\vdash \mathbf{0} : 0}\ \text{T-ZERO} \qquad \frac{\vdash P : n}{\vdash [P] : n+1}\ \text{T-LOC}$$

$$\frac{\vdash P : n}{\vdash \text{IN}.P : n}\ \text{T-IN} \qquad \frac{\vdash P : n \qquad \vdash Q : m}{\vdash P \mid Q : n+m}\ \text{T-PAR}$$

We reconsider the example process $P = \text{IN.IN.}[\mathbf{0}] \mid [[\mathbf{0}]]$ from Example 1. It holds that $\vdash P : 3$. After the reduction of P to $P' = [[[\mathbf{0}]]]$ the process still has the same type $\vdash P' : 3$. The property that types are preserved under reduction is called a subject reduction result and is a desirable feature of type systems.

Model Checking

The procedure of checking that a certain system satisfies a specific property is called model checking [55, 106]. The property that needs to be checked can often be formulated as a logical proposition. By using different logics it is possible to formulate different kinds of propositions. Temporal logics allow statements about timing aspects, while modal logics allow general statements about the development of processes. A modal logic [24,59] extends classical propositional logic to include operators expressing modality. A modal operator qualifies a statement, enabling modal logic to not only express truth of statements, but also the possibility that a statement is maybe true or may become true in the future. In this way, modal logics can be used to characterize various kinds of properties of systems. It allows, for example, to not only make statements about the current state of a system, but also about its development during runtime. For process calculi and logical formulas, the strict mathematical formulations allow for resolving model checking algorithmically. A finite state space ensures that the model checking algorithm always terminates.

Example 4 (IN-calculus: Modal logic)**.** *We define a modal logic for the* IN*-calculus. The formulas of the logic are defined as follows:*

$$\mathcal{A}, \mathcal{B} ::= \mathbf{0}, \neg\mathcal{A}, \mathcal{A} \wedge \mathcal{B}, \mathcal{A} \mid \mathcal{B}, [\mathcal{A}], \diamond\mathcal{A}.$$

We define the satisfaction relation via the reduction semantics of the calculus, where $\rightarrowtriangle^*$ *describes an arbitrary number of reduction steps:*

$$
\begin{array}{lcl}
P \vDash \mathbf{0} & \Leftrightarrow & P \equiv \mathbf{0} \\
P \vDash \neg A & \Leftrightarrow & P \nvDash \mathcal{A} \\
P \vDash \mathcal{A} \wedge \mathcal{B} & \Leftrightarrow & P \vDash \mathcal{A} \text{ and } P \vDash \mathcal{B} \\
P \vDash \mathcal{A} \mid \mathcal{B} & \Leftrightarrow & \exists P', P'' \text{ s.t. } P \equiv P' \mid P'' \text{ and } P' \vDash \mathcal{A}, P'' \vDash \mathcal{B} \\
P \vDash [\mathcal{A}] & \Leftrightarrow & P \equiv [P'] \text{ and } P' \vDash \mathcal{A} \\
P \vDash \diamond A & \Leftrightarrow & \exists P' \text{ s.t. } P \rightarrowtriangle^* P' \text{ and } P' \vDash \mathcal{A}.
\end{array}
$$

We reconsider the example process $P = \text{IN.IN.}[\mathbf{0}] \mid [[\mathbf{0}]]$ *from Example 1 and check if it satisfies the formula* $\mathcal{A} = \diamond[[[\mathbf{0}]]]$. *As* P *reduces to* $P' = [[[\mathbf{0}]]]$ *and* $P' \vDash [[[\mathbf{0}]]]$ *it holds that* $P \vDash \mathcal{A}$.

CHAPTER 3

The Calculus of Mobile Ambients

To formalize nested virtualization in cloud computing, notions of location mobility and nesting are essential. The calculus of mobile ambients focuses on processes executing at distributed locations and captures hierarchical nesting as well as mobility of explicit locations. This makes the ambient calculus a suitable choice as a starting point for the modeling of nested virtualization.

The calculus of mobile ambients was developed by Cardelli and Gordon [43] to model administrative domains for processes in distributed systems. Cardelli and Gordon found the inspiration for their calculus in the potential comprised by mobile computation via the Internet. Mobile computation, on the one hand, can stand for code that moves between devices, like apps, on the other hand, it can stand for computation that is carried out on mobile devices, like smart phones. Cardelli and Gordon aimed to describe both aspects of mobile computing in one single framework that included mobile processes, locations where mobile processes are executed, and the mobility of the locations themselves.

Even though the original paper was mostly concerned with formalizing the authorization needed to enter and exit an administrative domain, the creation of the ambient calculus went much further than this and defined a new paradigm of mobility where ambients are nested, processes are located inside ambients and ambients move under the control of the processes inside them. The calculus of mobile ambients innovated the area of mobile computing insofar as it allowed moving agents to be locations themselves and to contain data and running processes.

In the following we introduce the syntax and semantics of the basic mobile ambient calculus as well as a notion of weak bisimulation for mobile ambients on which we build in our research papers. We conclude this chapter with a discussion of work related to the calculus of mobile ambients.

	n	name	**Systems:**		
	x	variable	$M, N ::=$	$\mathbf{0}$	inactive system
Capabilities:				$M \mid N$	parallel composition
$C ::=$	*in* n	can enter n		$(\nu n)M$	restriction
	out n	can exit n		$n[P]$	ambient
	open n	can open n	**Processes:**		
Expressions:			$P, Q ::=$	$\mathbf{0}$	inactive process
$E, F ::=$	x	variable		$P \mid Q$	parallel composition
	C	capability		$(\nu n)P$	restriction
	$E.F$	path		$n[P]$	ambient
	ε	null		$!G.P$	replication
Guards:				$G.P$	prefixing
$G ::=$	E	expression		$\langle E \rangle$	async. output action
	(x)	input			

Table 3.1: Syntax of the ambient calculus.

3.1 Syntax and Semantics of Mobile Ambients

Ambients have been introduced to represent "administrative domains" for processes. In the calculus such domains consist of a process together with a name representing the location or domain where that process is running. Not only can ambients be nested, but the nesting structure can change dynamically. This is specified by prefixing a process with a *capability* C. The syntax in Table 3.1 is based on the formalization of Merro and Zappa Nardelli [110] and represents the basic calculus for mobile ambients, extended for asynchronous communication with the input action $(x).P$ and the asynchronous output action $\langle E \rangle$. The main difference compared to the original definition [43] lies in a separation of processes into two levels, as *processes* and *systems*. This distinction is used to simplify proofs of bisimulation.

Definition 1 (Syntax)**.** *The syntax of the calculus of mobile ambients, extended for asynchronous communication, is given by the grammar in Table 3.1.*

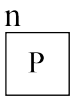

Figure 3.1: Graphical representation of the ambient $n[P]$.

In the following the processes of the calculus are described. The inactive process $\mathbf{0}$ does nothing. Parallel composition allows computation in P and Q to proceed simultaneously and is denoted by a binary operator $P \mid Q$ that is commutative and associative. The restriction operator $(\nu n)P$ creates a new and unique name in the

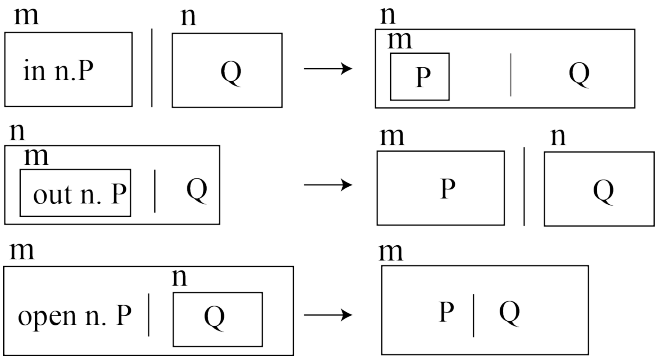

Figure 3.2: Graphical representation of the three basic capabilities.

scope of P. Finally, $n[P]$ denotes a process inside an ambient. For a better visualization of nesting structure mobile ambients can be represented as boxes, as can be seen in Fig. 3.1.

Processes can be prefixed $G.P$ with either an input action or a capability. There are three basic capabilities. The input capability *in* n indicates the willingness of a process (respectively its containing ambient) to enter an ambient named n, running in parallel outside, e.g.,

$$m[in\ n.P] \mid n[Q] \rightarrowtriangle n[m[P] \mid Q].$$

The output capability *out* n enables an ambient to leave its surrounding ambient n, e.g.,

$$n[m[out\ n.P] \mid Q] \rightarrowtriangle m[P] \mid n[Q].$$

The third basic capability *open* n allows to dissolve an ambient named n which is on the same level as the capability, thus, exposing its contents to the environment, e.g.,

$$m[open\ n.P \mid n[Q]] \rightarrowtriangle m[P \mid Q].$$

A visual representation of the changes in the nesting structure caused by the three basic capabilities can be seen in Fig. 3.2. Capabilities can be concatenated, thus forming a path. We omit trailing inactive processes, writing E instead of $E.\mathbf{0}$ and $n[]$ instead of $n[\mathbf{0}]$.

This syntax, as well as the semantics we consider, is based on [110] and largely unchanged compared to [43]. Note that Table 3.1, following [110], only allows replicated prefixing $!G.P$. Since replicated input in the π-calculus has the same expressive power as full replication [80] and recursion [115, 146], the replication of an ambient in the ambient calculus can be implemented with replicated prefixing and stuttering; i.e.

$$\begin{aligned} & m[] \mid k[!in\ m.out\ m.n[]] \\ \rightarrowtriangle & m[] \mid k[in\ m.out\ m.n[] \mid !in\ m.out\ m.n[]] \\ \rightarrowtriangle & m[k[out\ m.n[] \mid !in\ m.out\ m.n[]] \\ \rightarrowtriangle & m[] \mid k[n[] \mid !in\ m.out\ m.n[]] \end{aligned}$$

$P \equiv P$	(S-Refl)	$P \mid \mathbf{0} \equiv P$	(S-ZeroPar)
$P \equiv Q \Rightarrow Q \equiv P$	(S-Symm)	$(\nu n)\mathbf{0} \equiv \mathbf{0}$	(S-ZeroRes)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(S-Trans)	$!\mathbf{0} \equiv \mathbf{0}$	(S-ZeroRepl)
$P \mid Q \equiv Q \mid P$	(S-ParComm)	$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$	(S-Res)
$(P \mid Q) \mid R \equiv Q \mid (P \mid R)$	(S-Par-Assoc)	$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(S-Par)
$!P \equiv P \mid !P$	(S-Repl Par)	$P \equiv Q \Rightarrow !P \equiv !Q$	(S-Rep)
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(S-ResRes)	$P \equiv Q \Rightarrow n[P] \equiv n[Q]$	(S-Amb)
$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$ if $n \notin fn(P)$	(S-Res-Par)	$P \equiv Q \Rightarrow C.P \equiv C.Q$	(S-Cap)
$(\nu n)(m[P]) \equiv m[(\nu n)P]$ if $n \neq m$	(S-Res-Amb)		

Table 3.2: Structural congruence.

which results in the same outcome as a direct replication of the ambient n. We shall use $!n[]$ as an abbreviation for the stuttering bypass denoted above. Additionally, we use the following notational conventions. Parallel composition has the lowest precedence among the operators; e.g., for communication $(x).P \mid Q$ is read as $((x).P) \mid Q$. The process $E.F.P$ is read as $E.(F.P)$. For names, the ν-operator acts as a binder and the sets of free names fn and free variables of a process are defined as expected.

Definition 2 (Free names)**.** *The set of* free names *of a process, respectively of a capability, is inductively defined as:*

$$
\begin{aligned}
fn(\mathbf{0}) &\triangleq \emptyset & fn(x) &\triangleq \emptyset \\
fn(P \mid Q) &\triangleq fn(P) \cup fn(Q) & fn(n) &\triangleq \{n\} \\
fn((\nu n)P) &\triangleq fn(P) \setminus \{n\} & fn(in\ n) &\triangleq \{n\} \\
fn(!P) &\triangleq fn(P) & fn(out\ n) &\triangleq \{n\} \\
fn(n[P]) &\triangleq \{n\} \cup fn(P) & fn(open\ n) &\triangleq \{n\} \\
fn(C.P) &\triangleq fn(C) \cup fn(P) & fn(\varepsilon) &\triangleq \emptyset \\
fn((x).P) &\triangleq fn(P) & fn(C.C') &\triangleq fn(C) \cup fn(C') \\
fn(\langle C \rangle) &\triangleq fn(C)\ . &&
\end{aligned}
$$

The set of free variables x can be defined analogously, where the input prefix (x) is a binding occurrence for x.

The semantics is given as a reduction semantics which combines structural congruence with reduction rules. Structural congruence $P \equiv Q$ relates different syntactic representations of the same process and can be seen in Table 3.2. The reduction relation $P \rightarrowtriangle Q$, which describes the evolution of a process, is captured by the reduction rules in Table 3.3. In the case of communication we write $P\{x \leftarrow E\}$ for the substitution of the expression E for each free occurrence of the variable x in the process P.

$P \rightarrowtriangle Q \Rightarrow (\nu n)P \rightarrowtriangle (\nu n)Q$	(R-Res)
$P \rightarrowtriangle Q \Rightarrow P \mid R \rightarrowtriangle Q \mid R$	(R-Par)
$P \rightarrowtriangle Q \Rightarrow n[P] \rightarrowtriangle n[Q]$	(R-Amb)
$P' \equiv P, P \rightarrowtriangle Q, Q \equiv Q' \Rightarrow P' \rightarrowtriangle Q'$	(R-Red $\equiv$)
$n[in\ m.P \mid Q] \mid m[R] \rightarrowtriangle m[n[P \mid Q] \mid R]$	(R-In)
$m[n[out\ m.P \mid Q] \mid R] \rightarrowtriangle n[P \mid Q] \mid m[R]$	(R-Out)
$open\ n.P \mid n[Q] \rightarrowtriangle P \mid Q$	(R-Open)
$(x).P \mid \langle E \rangle \rightarrowtriangle P\{x \leftarrow E\}$	(R-Comm)

Table 3.3: Reduction rules.

3.2 Weak Bisimulation for Mobile Ambients

In order to compare different systems of ambients we make use of bisimulation. In this section we introduce a notion of weak bisimulation for mobile ambients. The following definitions establish the basis for our work on weak timed bisimulation for virtually timed ambients, which is presented in the research papers. This section is largely an exposition of the work of Merro and Zappa Nardelli [110].

A notion of weak bisimulation can be defined for the syntax given in Table 3.1. To define an appropriate notion of bisimulation, we need to revisit the operational semantics given above. The reduction semantics from Section 3.1 captures the behavior of a system considered *closed* or global, i.e., without interaction with its environment or with a surrounding context. Before defining a notion of bisimulation, we need to formalize an *open* version of the operational semantics, using a labelled transition relation instead of the reduction relation $\rightarrowtriangle$ used previously. To capture interaction with a surrounding *context* in the open setting, the transitions have to be *labelled* appropriately to capture the interaction with the environment. In our treatment of bisimulation, we omit communication; it can easily be added [110].

As arbitrary contexts can contain any kind of processes we add a placeholder $\circ$ to the syntax of mobile ambients which can be instantiated later in the bisimulation. The definition of the transition relation proceeds in two stages. Ultimately, we are interested in labelled transitions, where one distinguishes unobservable transitions, traditionally labelled by τ, and observable ones. This will be explained in detail later. In a preliminary step we define *pre*-actions or pre-transition in Table 3.4, as a technical stepping stone towards the transition relation afterwards.

The ambient syntax from Table 3.1 supports three basic prefixing operations as part of the capability syntax, omitting communication: entering, exiting, and opening. All three are reflected in the labels, but besides that, three additional labels for prefixing and pre-labelling are used, namely `enter_`n, `amb_`n, and `exit_`n. Thus, the actions labelled with π in Table 3.4 are elements of the set $\{$`in_`n, `out_`n, `open_`n, `enter_`n, `amb_`n, `exit_`$n\}$. Rule Pre-Pref of Table 3.4 deals with arbitrary capability prefixing in a standard manner. The rules for replicating a process, respectively a prefixed process works as expected. The next rule Pre-Enter deals ultimately via capability prefixes with *in* n. The capability prefix is translated into an enter-labelled transition. These

$$\frac{}{\pi.P \xrightarrow{\pi} P}\ \textsc{Pre-Pref} \qquad \frac{P \xrightarrow{\pi} Q}{!P \xrightarrow{\pi} Q \mid !P}\ \textsc{Pre-Repl}$$

$$\frac{P \xrightarrow{\texttt{in\_}n} P_1}{m[P] \xrightarrow{\texttt{enter\_}n} \langle m[P_1]\rangle \mathbf{0}}\ \textsc{Pre-Enter} \qquad \frac{P \xrightarrow{\texttt{out\_}n} P_1}{m[P] \xrightarrow{\texttt{exit\_}n} \langle m[P_1]\rangle \mathbf{0}}\ \textsc{Pre-exit}$$

$$\frac{}{n[P] \xrightarrow{\texttt{amb\_}n} \langle P\rangle \mathbf{0}}\ \textsc{Pre-Amb} \qquad \frac{P \xrightarrow{\pi} O \qquad n \notin fn(\pi)}{(\nu n)P \xrightarrow{\pi} (\nu n)O}\ \textsc{Pre-Res}$$

$$\frac{P \xrightarrow{\pi} O}{P \mid Q \xrightarrow{\pi} O \mid Q}\ \textsc{Pre-Par}_1$$

Table 3.4: Pre-actions

pre-transitions are not directly dealing with transitions of a system; they are rather introduced to formulate the real transitions later. The post-state of the pre-transitions are likewise not necessarily other processes or ambients; instead a new intermediate form $(\nu\widetilde{m})\langle P\rangle Q$ is introduced, know as *concretion*. It consists of two processes together with an indication of ν-bound names. In that concretion, P is the "relevant part" of the process, i.e., the process involved in entering or exiting an ambient, whereas Q represents the rest of the process. Processes together with concretions are also referred to as *outcomes* and are represented by O, as opposed to P, Q, ... representing processes. In PRE-ENTER, no bindings are involved and the ambient m, signalling its willingness of entering ambient n, is "temporarily stored" in the concretion $\langle m[P_1]\rangle \mathbf{0}$. Rule PRE-EXIT works dual to the one for entering. Ambients n advertise their existence doing an $\texttt{amb\_}n$-step (see PRE-AMB), using again a concretion as outcome. The remaining two rules are structural in that they treat labelling in connection with ν-binders and parallel composition. We omit rule $\textsc{Pre-Par}_2$ which is symmetric to $\textsc{Pre-Par}_1$.

Unlike the preliminary transitions just discussed, the next two tables formalize labelled transitions of "real" processes or systems. In particular, the transitions in the conclusion of the rules will not involve any concretions as opposed to the premises, which use the rules for pre-transitions from Table 3.4. As mentioned, the tables split the transition relation into internal τ-steps, and observable steps. Note that the actions a process or ambient can do — entering, exiting, and opening — are covered by respective rules in both Tables 3.5 and 3.6, depending on whether the action is dealt with internally or via environment interaction. Especially the τ-transitions dealing with the basic actions of processes resemble the corresponding steps for the reduction relation by rules R-OPEN, R-IN, and R-OUT from Table 3.3.

The rules for τ-transitions are mostly straightforward. A process indicating its

$$\frac{(fn(k[P_1]) \cup fn(P_2)) \cap \{\widetilde{q}\} = \emptyset = ((fn(Q_1) \cup fn(Q_2)) \cap \{\widetilde{p}\} \quad P \xrightarrow{\texttt{enter\_}n} (\nu\widetilde{p})\langle k[P_1]\rangle P_2 \qquad Q \xrightarrow{\texttt{amb\_}n} (\nu\widetilde{q})\langle Q_1\rangle Q_2}{P \mid Q \xrightarrow{\tau} (\nu\widetilde{p})(\nu\widetilde{q})(n[k[P_1] \mid Q_1] \mid P_2 \mid Q_2)} \text{ R-TauEnter}_1$$

$$\frac{P \xrightarrow{\texttt{exit\_}n} (\nu\widetilde{m})\langle k[P_1]\rangle P_2}{n[P] \xrightarrow{\tau} (\nu\widetilde{m})(k[P_1] \mid n[P_2])} \text{ R-TauExit}$$

$$\frac{P \xrightarrow{\texttt{open\_}n} P_1 \qquad Q \xrightarrow{\texttt{amb\_}n} (\nu\widetilde{m})\langle Q_1\rangle Q_2}{P \mid Q \xrightarrow{\tau} (\nu\widetilde{m})\,(Q_1 \mid Q_2) \mid P_1} \text{ R-Tau-Open}_1$$

$$\frac{P \xrightarrow{\tau} Q}{n[P] \xrightarrow{\tau} n[Q]} \text{ R-TauAmb} \qquad \frac{P \xrightarrow{\tau} Q}{(\nu n)\,P \xrightarrow{\tau} (\nu n)\,Q} \text{ R-TauRes} \qquad \frac{P \xrightarrow{\tau} P'}{P \mid Q \xrightarrow{\tau} P' \mid Q} \text{ R-TauPar}_1$$

Table 3.5: Silent actions

intention to enter an ambient n paired with such an ambient, which in turn advertises its presence by an $\texttt{amb\_}n$-transition, synchronizes into a τ-transition and the resulting post-configuration contains the post-state of P having entered the target ambient named n (see R-TauEnter$_1$, a symmetric rule R-TauEnter$_2$ is omitted). The premise concerning the use of names is needed to avoid capture of names in the scope of $(\nu\widetilde{p})$ and $(\nu\widetilde{q})$ in the process after the step. Doing an exit-transition from a given ambient n results in a τ-step, as it again involves no outside interaction (see R-TauExit): the exiting ambient named k leaves its immediately surrounding ambient named n. As for opening an ambient: If P is able to perform the capability *open* n and the partner process Q contains an ambient n at the top level, then the rule R-Tau-Open$_1$ leads to a τ-step reducing P and opening and dissolving ambient n in Q. The next two rules R-TauAmb and R-TauRes specify that τ-steps can occur unhindered inside an ambient respectively underneath a ν-binder. Rule rule R-TauPar$_1$ is standard, as well, and expresses the interleaving semantics for silent steps. The symmetric rule R-TauPar$_2$ is omitted.

As the ones for τ-steps, the rules for *environment* actions or transitions mentions pre-actions or pre-transitions in their premises. The transition relation serves to formulate an appropriate notion of bisimulation, which we later will generalize to our timed setting. The observable labels are, basically, of three different kinds, $k.\texttt{enter\_}n$, $k.\texttt{exit\_}n$, and $k.\texttt{open\_}n$, corresponding to the three basic ambient capabilities. However, the label of entering has additionally a dual counterpart $k.\overline{\texttt{enter}}\texttt{\_}n$. Compared to the labels used in the pre-transitions, both the "source" and the "target" ambient are part of the label. The entering and exiting labels also exist in an "anonymous" variant, where the source ambient executing the corresponding capability, remains hidden. This is represented by $*$. With these labels, the transitions of the open system contain

$$\frac{P \xrightarrow{\texttt{enter\_}n} (\nu\widetilde{m})\langle k[P_1]\rangle P_2 \quad k \notin \widetilde{m}}{P \xrightarrow{k.\texttt{enter\_}n} \nu(\widetilde{m})(n[k[P_1] \mid \circ] \mid P_2)} \text{ T-ENTER}$$

$$\frac{P \xrightarrow{\texttt{amb\_}n} (\nu\widetilde{m})\langle P_1\rangle P_2 \quad k \notin \widetilde{m}}{P \xrightarrow{n.\overline{\texttt{enter}}\_k} \nu(\widetilde{m})(n[P_1 \mid k[\circ]] \mid P_2)} \text{ T-COENTER}$$

$$\frac{P \xrightarrow{\texttt{enter\_}n} (\nu\widetilde{m})\langle k[P_1]\rangle P_2 \quad k \in \widetilde{m}}{P \xrightarrow{*.\texttt{enter\_}n} \nu(\widetilde{m})(n[k[P_1] \mid \circ] \mid P_2)} \text{ T-ENTERSHH}$$

$$\frac{P \xrightarrow{\texttt{exit\_}n} (\nu\widetilde{m})\langle k[P_1]\rangle P_2 \quad k \notin \widetilde{m}}{P \xrightarrow{k.\texttt{exit\_}n} (\nu\widetilde{m})(k[P_1] \mid n[\circ \mid P_2])} \text{ T-EXIT}$$

$$\frac{P \xrightarrow{\texttt{exit\_}n} (\nu\widetilde{m})\langle P_1\rangle P_2 \quad k \in \widetilde{m}}{P \xrightarrow{*.\texttt{exit\_}n} (\nu\widetilde{m})(n[k[P_1] \mid \circ] \mid P_2)} \text{ T-EXITSHH}$$

$$\frac{P \xrightarrow{\texttt{amb\_}n} (\nu\widetilde{m})\langle P_1\rangle P_2}{P \xrightarrow{k.\texttt{open\_}n} k[\circ \mid (\nu\widetilde{m})(P_1 \mid P_2)]} \text{ T-OPEN}$$

Table 3.6: Environment actions

relevant information of the system-environment interaction as far as ambient-behavior is concerned, like the names of the involved ambients and the nature of the interaction. Not *all* information, however, is recorded in the labels. For instance, when an external process or ambient k requests to enter ambient n, captured by a transition labelled $n.\overline{\texttt{enter}}\_k$, then conceptually after the step, the requesting ambient k will have entered the local ambient n. However, the ambient itself is not mentioned on the labels, as only an ambient's or process's interaction is considered visible, not the ambient or process itself. To represent in this and similar situations in the post-state an *arbitrary process*, we extend the syntax slightly introducing $\circ$ as placeholder or "special variable" for arbitrary processes. This placeholder will also be crucial later when formulating the used notion of bisimulation, again capturing the openness of the system and the arbitrariness of part of the considered processes, after performing the different environment transitions.

Definition 3 (Extended syntax)**.** *We extend the syntax of mobile ambients by adding the production* $P ::= \circ$ *and call the grammar of Table 3.1 extended in this way with* $\circ$ *the* extended syntax.

With this extension, the environment interaction is captured by the rules of Table 3.6.

$(\nu\widetilde{m})(m[in\ n.P \mid Q] \mid M), m \in \widetilde{m} \xrightarrow{*.\mathtt{enter}\_n} (\nu\widetilde{m})(n[m[P \mid Q] \mid \circ] \mid M)$	(Enter Shh)
$(\nu\widetilde{m})(k[in\ n.P \mid Q] \mid M), k \notin \widetilde{m} \xrightarrow{k.\mathtt{enter}\_n} (\nu\widetilde{m})(n[k[P \mid Q] \mid \circ] \mid M)$	(Enter)
$(\nu\widetilde{m})(m[out\ n.P \mid Q] \mid M), m \in \widetilde{m} \xrightarrow{*.\mathtt{exit}\_n} (\nu\widetilde{m})(m[P \mid Q] \mid n[M \mid \circ])$	(Exit Shh)
$(\nu\widetilde{m})(k[out\ n.P \mid Q] \mid M), k \notin \widetilde{m} \xrightarrow{k.\mathtt{exit}\_n} (\nu\widetilde{m})(k[P \mid Q] \mid n[M \mid \circ])$	(Exit)
$(\nu\widetilde{m})(k[P] \mid M), k \notin \widetilde{m} \xrightarrow{k.\overline{\mathtt{enter}}\_n} (\nu\widetilde{m})(k[n[\circ] \mid P] \mid M)$	(Co-Enter)
$(\nu\widetilde{m})(k[P] \mid M) \xrightarrow{n.\mathtt{open}\_k} n[\circ \mid (\nu\widetilde{m})(P \mid M)]$	(Open)

Table 3.7: Env-actions.

Unlike the τ-rule dealing with component internal transitions, the labelled transitions of environment interaction from Table 3.6 consider interactions with ambients from the outside. As its previous silent counterparts R-TauEnter$_i$, T-Enter deals with a process indicating its intention to enter an ambient named n. The previous one dealt with that wish locally, using a τ-transition. As in the local case, the process P indicates its wish to enter n via the corresponding pre-transition. In absence of a specified partner ambient to enter into, the post-configuration in T-Enter makes use of the placeholder $\circ$. Rule T-CoEnter is the dual one to T-Enter . Both parts can be seen as one half each of R-TauEnter covering the internal case of entering. In T-CoEnter. an ambient n is willing to being entered by an ambient k from the outside, and $\circ$ in the configuration afterwards again represents the fact that any arbitrary process, whose exact value is not recorded in the label, may be contained in the ambient k. Rule T-EnterShh is the "anonymous" version of entering as in T-Enter. The difference to the previous interaction is that the name k of the internal ambient indicating its entering-wish is hidden by the ν-binder. Therefore, its name k is suppressed in the transition label in the conclusion. Note that rule T-CoEnter has no anonymous counterpart. Rule T-Exit, corresponding to the reduction rule R-Exit, formalizes if an ambient leaves its immediately surrounding ambient, named n in that rule. Thus, it ultimately models the execution of an *out* n-capability. Note the close similarity to the internal rule R-TauExit which shares the same premise. The previous τ-rule was formulated for the ambient n which is the one process P intends to exit. In that situation, both the ambient which wants to exit n as well as the surrounding ambient being exited were part of the configuration for which the rule was formulated. That made the step internal, i.e., a step labelled by τ. The situation for the open system in T-Exit here is different. The surrounding ambient n, the one being exited, is *not* part of the configuration prior to the step, which makes it an open system. Thus, the wish of k to leave n is indicated on the label. Secondly, in absence of ambient n before the transition, $\circ$ is used to represent the unknown part of that ambient in the post-state. T-ExitShh is again the variant of the version without "Shh", where the name of the ambient executing the command is suppressed as it occurs hidden behind a ν-binder. The labelled transition for opening an ambient in rule T-Open is the open-system version of the previous rule R-TauOpen. Unlike in the rule R-TauOpen, here the partner ambient k, the one which is responsible to open ambient n, is absent, but its intention is indicated in the label of the transition. Again,

in the post-configuration, its unknown internals are represented by the placeholder $\circ$.

Definition 4 (Untimed labelled transitions)**.** *The* env-actions *of the labelled transition system are defined in Table 3.7. Unobservable τ-actions, which model the internal evolution of a process and can not be seen from the outside, are defined as in [110].*

In the rules (Enter) and (Exit) an ambient k enters, respectively exits, from an ambient n provided by the environment. The rules (Enter Shh) and (Exit Shh) model the same behavior for ambients with private names. In the rule (Co-Enter) an ambient n provided by the environment enters an ambient k of the process. In the rule (Open), the environment provides an ambient n in which the ambient k of the process is opened. The reduction semantics of a process can be encoded in the labelled transition system, because a reduction step can be seen as an interaction with an empty context.

Note that the transition semantics contains $\circ$ as placeholder for an arbitrary process. It can be seen as a special variable representing arbitrary processes. The process $P := \circ$ must be instantiated in the bisimulation. The replacement of that variable by a process is written as $P \bullet Q$ and defined as expected.

Definition 5 (Substitution)**.** *Let P and Q be processes over the extended syntax. Let R be a process. The capture-avoiding* substitution *of R for the occurrences of $\circ$ in a process is defined as follows:*

$$\begin{aligned}
\mathbf{0}\bullet R &= \mathbf{0} & n[P]\bullet R &= n[P\bullet R] \\
\circ\bullet R &= R & !C.P\bullet R &= !C.(P\bullet R) \\
(P \mid Q)\bullet R &= (P\bullet R) \mid (Q\bullet R) & (\nu n)P\bullet R &= (\nu n)(P\bullet R) \text{ if } n \notin fn(R) \\
C.P\bullet R &= C.(P\bullet R)\ . &&
\end{aligned}$$

We are interested in bisimulations that abstract from τ-actions and introduce the notion of *weak actions*: let $\Longrightarrow$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$, let $\overset{\alpha}{\Longrightarrow}$ denote $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$, and let $\overset{\hat{\alpha}}{\Longrightarrow}$ denote $\Longrightarrow$ if $\alpha = \tau$ and $\overset{\alpha}{\Longrightarrow}$ otherwise.

Definition 6 (Weak bisimulation)**.** *A symmetric relation $\mathcal{R}$ over systems is a* weak bisimulation *if $M\ \mathcal{R}\ N$ implies:*

- *if $M \xrightarrow{\alpha} M'$, $\alpha \in \{\tau, k.\texttt{enter\_}n, k.\texttt{exit\_}n, k.\overline{\texttt{enter}}\texttt{\_}n, n.\texttt{open\_}k\}$, then there is a system N' such that $N \overset{\hat{\alpha}}{\Longrightarrow} N'$ and for all processes P it holds that $M'\bullet P\ \ \mathcal{R}\ \ N'\bullet P$;*
- *if $M \xrightarrow{*.\texttt{enter\_}n} M'$ then there is a system N' such that $N \mid n[\circ] \Longrightarrow N'$ and for all processes P it holds that $M'\bullet P\ \ \mathcal{R}\ \ N'\bullet P$;*
- *if $M \xrightarrow{*.\texttt{exit\_}n} M'$ then there is a system N' such that $n[\circ \mid N] \Longrightarrow N'$ and for all processes P it holds that $M'\bullet P\ \ \mathcal{R}\ \ N'\bullet P$.*

Systems M and N are weakly bisimilar*, written $M \approx N$, if $M\ \mathcal{R}\ N$ for some weak bisimulation $\mathcal{R}$.*

3.3 Mobile Ambients in a Larger Context

Since its inception [43] the calculus of mobile ambients has spread far and wide. It has inspired papers on mobile computation and has been applied in various directions of computer science, from better design approaches for mobile agent applications [160] to providing information about security issues related to mobility [123], formal analysis of sensor networks [61] and the description of network protocols [10]. Furthermore, it has not only found acceptance in computer science but was utilized amongst other disciplines also in engineering [92] and biochemistry [141]. The following discussion of literature is by no means complete, but is meant to give a broad overview over the different research areas that have surrounded the ambient calculus in the last two decades, in order to place our research on virtually timed ambients in this larger context.

Virtualization. Gordon proposed a simple formalism for virtualization loosely based on mobile ambients [74] and the calculus of mobile ambients itself has been used as a framework for the specification of the migration of virtual machines [84]. This approach focuses on security, uses mobile ambients as firewalls, and does not consider time and resources. In comparison the calculus of virtually timed ambients [88, 89] stays closer to the syntax of the original mobile ambient calculus than Gordon's formalism, while at the same time including notions of time and explicit resource provisioning which allow interesting aspects of virtualization to be expressed.

Bisimulations. Cardelli and Gordon defined a labeled transition system for their mobile ambients [45], but no bisimulation. A bisimulation relation for a restricted version of mobile ambients, called Mobile Safe Ambients [102], is defined in [109] and provides the basis for later work. Barbed congruence for the same fragment of mobile ambients is defined in [157]. It is shown in [67] that name matching reduction barbed congruence and bisimulation coincide in the π-calculus. A bisimulation relation with contextual labels for the ambient calculus is defined in [120], but this approach is not suitable for providing a simple proof method. A simulation-based faster-than preorder is introduced in [103]; this preorder is related to our notion of time relaxation but the faster-than preorder allows a process to delay by at most one time unit. A labelled bisimulation for mobile ambients is defined by Merro and Nardelli [110], who prove that this bisimulation is equivalent to reduction barbed congruence and develop up-to proof techniques. The weak timed bisimulation defined in our paper [89] is a conservative extension of this approach, which is extended further in our later work regarding resource-awareness, using notions of context bisimulation developed in [143, 146].

Fragments. In order to address concerns raised by the original ambient calculus, a variety of fragments and variants of the ambient calculus have been defined and studied. One of the first concerns that was raised regarding the basic ambient calculus is the security risk that the *open* n capability poses to a system. When an ambient is

opened all its contents are released regardless what they are. Without further rules or restrictions this can constitute a threat to security. This issue has been addressed in various ways. In Mobile Safe Ambients [102] the movement capabilities are modified to control interference and Secure Safe Ambients [33] are a typed variant of this calculus. Boxed Mobile Ambients [34, 35] solve the issue by just dropping the open capability completely and defining new primitives for process communication. Monitoring and coordination are emphasized in Guarded Boxed Ambients [63], an enhancement of boxed ambients with guardians. The expressiveness of the ambient calculus without the open capability is studied in [25]. The Push and Pull Ambient Calculus changes the perspective on the capabilities and considers ambients to be passively pushed away and pulled in by other ambients instead of actively entering and exiting [134]. Other modifications are the addition of fairness principles to safeguard the interactions of the ambients in Fair Ambients [68] and the addition of context-expressions in the Calculus of Context-aware Ambients to model context-aware systems [150].

Type Systems. A type system for the originally untyped ambient calculus was defined in [41] and refined later [39,40]; this type system is mainly concerned with the use of groups to control communication and mobility. For communication, a basic type of an ambient captures the kind of messages that can be exchanged within. For mobility, the type system controls which ambients can enter. In a more traditional setting of sequential languages, types are often enriched with effects to capture the aspects of computation which are not purely functional. In process algebra, session types have been used to capture communication in the π-calculus. Orchard and Yoshida have shown that effects and session types are similar concepts as they can be expressed in terms of each other [126]. Session types have been defined for boxed ambients in [69] and behavioral effects for the ambient calculus in [12], where the original communication types by Cardelli and Gordon are enhanced by movement behavior. This is captured with traces, the flow-sensitivity hereby results from the copying of the capabilities in the type.

To capture how a computation depends on an environment instead of how the computation affects it, Petricek, Orchard and Mycroft suggest the term coeffect as a notion of context-dependent computation [132, 133]. Dual to effects, which can be modeled monadically, the semantics of coeffects is provided by indexed comonads [93, 156]. We use coeffects in our type system for virtually timed ambients to control time and resources. An approach to control timing via types can be found in [21], which develops types and typed timers for the timed π-calculus. Another approach to resource control without coeffects can be found in [78], which proposes a type system to restrict resource access for the distributed π-calculus. In [154] a type system for resource control for a fragment of the mobile ambients is defined by adding capacity and weight to communication types for controlled ambients. Simplified non-modifiable mobile ambients with resources, and types to control migration and resource distribution are proposed in [71]. Another fragment of the ambient calculus, finite control ambients with only finite parallel composition, are covered in [50]. This fragement was used to develop a model checking algorithm to admit automatic verification via state-space exploration.

Here the types are a bound to the number of allowed active outputs in an ambient.

Modal Logic and Model Checking. Modal logic for mobile ambients was introduced to describe properties of spatial configuration and mobile computation [42] for a fragment of mobile ambients without replication and restriction on names. The extensionality and intensionality of this logic is studied in [144]. The complexity of model checking for mobile ambients is investigated in [49], and shown to be PSPACE-complete. After Cardelli and Gordon's work on logical properties for name restriction [44], the model checker algorithm was extended for private names [51] while preserving decidability and the complexity of the original fragment. Furthermore, 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, our logic and model checker for virtually timed ambients is based on the original fragment from [42]. The modal operators with restrictions on timing in our implementation of modal logic for virtually timed ambients borrows ideas from metric temporal logic [96, 127, 128].

The Process Analysis Toolkit (PAT) [152] has been used to specify processes in the ambient calculus as well as properties in modal logic [153], 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 [6]. Here 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. Meanwhile, the analysis of temporal modalities are handled using the NuSMV model checker. A similar approach is used in [155] to specify security policies of mobile networks. In contrast to our work, none of the above model checkers consider notions of time or resources.

Implementations and Analyses. Mobile ambients have been investigated from different angles. The semantics of the calculus has been investigated in different ways [62, 75, 76] and the expressiveness has been studied for pure mobile ambients without communication [36] as well as for different fragments [104, 135]. The relation between the ambient calculus and the π-calculus has been investigated thoroughly [31, 32, 54]. And not only have algorithms been introduced to analyse nesting [28] but there has also been built a Java based graphical tool for the analysis of nesting and information leakage in mobile agent specifications [27]. The operational reduction rules for mobile ambients as well as a type system have been implemented in Maude [142]. In contrast, the implementation we present for virtually timed ambients focuses on capturing the timed reduction rules of the calculus as well as the modal formulas to define a model checker. Our implementation exploits the low representational distance which distinguishes the Maude system [111, 125].

CHAPTER 4

List of Research Papers

The research contributions of this thesis are presented in five papers which are briefly summarized in this chapter. The full papers can be found in Part II of this thesis, with mostly the same content as in their original publications. Note that all papers have been reformatted to fit the structure of this thesis and that spelling mistakes as well as small errors have been fixed. For reasons of completeness the second, third and fourth paper have been extended with additional explanations and proofs that were not part of the original publication. Following an established convention in theoretical computer science, the authors on each paper are listed in alphabetical order. We shortly present additional publications including extended abstracts and technical reports in the last section of this chapter.

The calculus that forms the basis for the work in this thesis is the calculus of mobile ambients by Cardelli and Gordon. It is described in Chapter 3. In the first paper we extend this calculus with time and resources, define a notion of bisimulation and perform a thorough theoretical investigation of the subject. After the publication of the first and before the publication of the second paper we adjusted the notations and changed the naming of the local clocks to schedulers in order to better describe their role in the calculus. Moreover, we changed the distribution strategy for time slices from a lazy round-robin like strategy, which emits all possible time slices for a round only after the number of input time slices is complete, to an eager round-based distribution strategy, which emits a time slice as soon as possible. The second paper provides the calculus with a type system for statical analysis, while the third paper enhances it with modal logic and provides a prototype implementation in the Maude system for rewriting logic. The fourth paper makes use of these foundations to built a calculus which includes resource-awareness, allowing systems to react dynamically to changes during runtime. The fifth paper presents the calculus of virtually timed ambients together with its prototype implementation from an applied point of view as an analysis framework for virtualization in cloud computing.

4.1 Paper 1: Virtually Timed Ambients: A calculus of nested virtualization

Authors Einar Broch Johnsen, Martin Steffen, Johanna Beate Stumpf

Publication Journal of Logical and Algebraic Methods in Programming, volume 94, pp 109-127, 2018

Summary To study the effects of nested virtualization, we first define the *calculus of virtually timed ambients*, to the best of our knowledge the first process algebra capturing notions of virtual time and resource provisioning for nested virtualization. Furthermore, we define *weak timed bisimulation* for virtually timed ambients, and show that weak timed bisimulation is equivalent to reduction barbed congruence with time. Lastly, we define *time relaxation* for virtually timed ambients as a simulation relation allowing deviation by a bounded amount of time.

Abstract nested virtualization enables a virtual machine, which is a software layer representing an execution environment, to be placed inside another virtual machine. Nested virtual machines form a location hierarchy where virtual machines at every level in the hierarchy compete with other processes at that level for processing time. With nested virtualization, the computing power of a virtual machine depends on its position in this hierarchy and may change if the virtual machine moves. We introduce the calculus of virtually timed ambients, a formal model of hierarchical locations for execution with explicit resource provisioning, motivated by these effects of nested virtualization. The calculus of virtually timed ambients is based on the calculus of mobile ambients. Mobile ambients are located processes, arranged in a hierarchy which may change dynamically. Interpreting the location as a place 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, but, on the other hand, on the placement or deployment of the component in the hierarchical ambient structure. Resource provisioning in this model is based on virtual time slices as a local resource. To reason about timed behavior in this setting we make use of bisimulation. We define a notion of weak bisimulation for virtually timed ambients as an extension of the formalization for mobile ambients by including the timed behavior and show that weak timed bisimulation is a congruence.

The timed behavior of a system depends on the kind of requests that are made, the speeds of the virtually timed ambients and the way requests are distributed to virtual machines. Thus, it is possible that two virtually timed systems behave the same for a certain type of requests but not for others, or that a system behaves equivalently to another except for needing some extra time slices per executed request. For this reason we define simulation with time relaxation to express that a system is slower than another system up to a given time bound.

4.2 Paper 2: Assumption Commitment Types for Virtually Timed Ambients

Authors Einar Broch Johnsen, Martin Steffen, Johanna Beate Stumpf

Publication Submitted to Logical Methods In Computer Science (under review).

Summary This paper introduces a *type system for resource management* in the context of nested virtualization. With nested virtualization, virtual machines compete with other processes for the resources of their host environment in order to provision their own processes, which could again be virtual machines. The calculus of virtually timed ambients formalizes such resource provisioning, extending the capabilities of mobile ambients to model the dynamic creation, migration, and destruction of virtual machines. Compared to earlier papers we work here with an *eager distribution strategy* for time slices and *local clocks* are renamed to *local schedulers* to better describe their role in the calculus. The change of the scheduling strategy turned out beneficial in the analysis in that it allowed a cleaner formulation of the assumption-commitment rules and the subsumption properties concerning the resource effects.

The proposed type system for virtually timed ambients analyzes the timed behavior in terms of movement and resource consumption of a given system. Statically estimating the timed behavior is complicated because the placement of an ambient in the system hierarchy influences its resource consumption, and moving inside the hierarchy changes its virtual speed. The concept of this type system is loosely based on Cardelli, Ghelli, and Gordon's movement control types for mobile ambients. However, its purpose is quite different, and therefore the technical formulation is different, too. Our type system uses assumptions about the outside of a virtually timed ambient to guarantee resource provisioning on the inside, enabling static checking of timing and resource constraints for ambients. We prove *subject reduction and progress* for well-typed virtually timed ambients, expressing that the upper bounds on resource needs are preserved by reduction and that processes will not run out of resources. The results are given for a non-standard assumption-commitment setting in an operational framework. The type system further provides reusable properties as it supports abstraction and the results would also hold for other operational accounts of fair scheduling strategies. It holds that the type system supports subsumption, which allows relating different types, for example weaker types, to each other.

4.3 Paper 3: Checking Modal Contracts for Virtually Timed Ambients

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

Publication Proceedings of the 15th International Colloquium on Theoretical Aspects of Computing (ICTAC 2018)

Summary 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. This paper introduces a modal logic for virtually timed ambients with notions from metric temporal logic, enabling us to define timed behavior and resource consumption of a system as modal logic properties of processes.

The main contribution of this paper is the introduction of *modal logic* for virtually timed ambients by means of combining modal logic for mobile ambients with notions of metric temporal logic in order to capture the special features of virtual time and resource provisioning in virtually timed ambients. We further show that the resulting logic is a *conservative extension* of the modal logic for the ambient calculus, preserving satisfiability and define a *model checking algorithm* for this modal logic, and develop a prototype implementation in Maude.

Modal logic for virtually timed ambients enables us 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. Thus, the proposed modal logic supports quality of service 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, to prove that a process satisfies a formula, and its prototype implementation in the Maude system for rewriting logic. 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. This leads to a close correlation between the formal definition of the satisfaction relations in the modal logic and the Maude specification. As satisfiability conditions of the modal logic can be compactly represented, we obtain a compact and flexible model checker which stays close to its mathematical formulation.

4.4 Paper 4: Resource-Aware Virtually Timed Ambients

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

Publication Proceedings of the 14th International Conference on integrated Formal Methods (iFM 2018)

Summary This paper introduces resource-awareness for virtually timed ambients, which not only allows to model dynamically changing systems of nested virtualization, but enables the modeled processes to actively query the system about the resources necessary for a task and to reconfigure depending on these queries. Based on the local load situation, a virtually timed system can for example decide to trigger the creation of a new virtual machine, thus enabling horizontal scaling for virtually timed ambients.

We define and discuss the calculus of *resource-aware virtually timed ambients* as well as *weak timed context bisimulation* for resource-aware virtually timed ambients. Furthermore, a *case study* of dynamic auto scaling on Amazon EC2 modeled in a prototype implementation of our calculus in the Maude system shows the feasibility of virtually timed ambients as a modeling language for cloud computing. To the best of our knowledge, this is the first implementation of resource-awareness for mobile ambients in rewriting logic.

Virtually timed ambients is a calculus which models timing aspects and resource consumption for virtual machines, as well as the hierarchical structure of systems with nested virtualization. This structure may change dynamically to support load-balancing, to move processes, or to re-provision virtual machine resources. To support resource-awareness, the paper technically extends virtually timed ambients with properties of context aware ambients, namely context-guarded actions, thus defining resource-awareness. Resource-aware processes have to fulfill a context requirement κ before they can continue to execute. Here κ ranges over context expressions, which are essentially modal formulas. We give a formal semantics for the extension and provide an appropriate formalization of bisimulation for comparing equivalent resource-aware virtually timed systems. The given notion of bisimulation makes use of contexts, which enables us to establish that this bisimulation relation is a congruence and coincides with reduction barbed congruence over the given systems.

The calculus is implemented in Maude, and illustrated by a case study based on the Auto Scaling User Guide by Amazon Web Services. The case study presents a cloud implementation in the calculus of resource-aware virtually timed ambients that allows a user to dynamically scale the number of Amazon EC2 instances available to handle the load for a given application.

4.5 Paper 5: An Analysis Framework for Virtualization

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

Publication Proceedings of the Norsk Informatikkonferanse (NIK 2018)

Summary We explore the calculus of virtually timed ambients as a tool for the analysis of virtualization in cloud computing. The main contributions of this paper are the presentation of the *virtually timed ambient calculus* and the corresponding modal logic as framework for the modeling of virtualization in cloud computing, a cloud library for *model implementations* in the rewriting logic system *Maude* and *examples and analysis* of cloud models developed in Maude.

Virtually timed ambients is a calculus of explicit resource provisioning, based on the well-known calculus of mobile ambients. 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. To analyze models of virtualization we introduce the calculus and its corresponding modal logic and utilize a prototype implementation of the calculus in rewriting logic.

In order to use the implementation of virtually timed ambients as a modeling language for cloud computing we develop a cloud library, containing important elements of cloud architecture which can be put together according to a modular principle. A cloud model in this implementation consists of a system containing a cloud ambient and several applications or data packages. The cloud includes a load balancing or scaling process, depending on the chosen load balancing strategy, as well as virtual machines. The applications and data packages enter the cloud in order to be executed or stored. We present three different exemplary models of cloud computing deploying three different ways of load balancing or scaling. In each case we simulate load on the machines and consider one special data package to survey the satisfaction of certain quality of service statements by the model. The quality of service statements are given as modal formulas and are verified using the model checker implementation in Maude.

4.6 Additional Publications

This section lists additional publications, which are not directly included in the thesis. The publications are related to the stated research goal and mostly correspond to preliminary versions of work presented in this thesis.

4.6.1. A Calculus of Virtually Timed Ambients

Authors: Einar Broch Johnsen, Martin Steffen, Johanna Beate Stumpf
Publication: UiO Technical Report TR-456, October 2016, ISBN 978-82-7368-421-9
Notes: Preparatory work for Paper 1.

4.6.2. A Calculus of Virtually Timed Ambients

Authors: Einar Broch Johnsen, Martin Steffen, Johanna Beate Stumpf
Publication: Postproceedings of selected contributions to the 23rd International Workshop on Algebraic Development Techniques (WADT 2016), 2017, editors: Phillip James and Markus Roggenbach
Notes: Can be considered a short version of Paper 1.

4.6.3. A Calculus of Virtually Timed Ambients

Authors: Einar Broch Johnsen, Martin Steffen, Johanna Beate Stumpf
Publication: Proceedings of the 28th Nordic Workshop on Programming Theory (NWPT'16), editors: Kim G. Larsen and Jiří Srba
Notes: Extended abstract of Paper 1.

4.6.4. Virtually Timed Ambients for Cloud Computing

Authors: Johanna Beate Stumpf
Publication: Sirius General Assembly, Poster, 23.05.2017, Oslo
Notes: Poster presentation regarding the research project.

4.6.5. Virtually Timed Ambients: Formalisation and Analysis

Authors: Johanna Beate Stumpf, Einar Broch Johnsen, Martin Steffen
Publication: Proceedings of the PhD Symposium at iFM'17 on Formal Methods: Algorithms, Tools and Applications (PhD-iFM'17), editors: Erika Abraham and Silvia Lizeth Tapia Tarifa, UiO Technical Report TR-470, September 2017, ISBN 978-82-7368-435-6
Notes: Extended abstract presenting the research project.

4.6.6. Assumption Commitment Types for Resource Management in Virtually Timed Ambients

Authors: Einar Broch Johnsen, Martin Steffen, Johanna Beate Stumpf
Publication: UiO Technical Report TR-472, October 2017, ISBN 978-82-7368-437-0
Notes: Preparatory work for Paper 2.

CHAPTER 5

Discussion and Conclusion

In the following we discuss the contributions of this thesis and relate them to the research questions stated in Chapter 1.2. We conclude with an outlook on prospective research that can be initiated by the results presented in this thesis.

5.1 Summary of Contributions

This thesis introduces the calculus of *virtually timed ambients* in Chapter 6, a formal model of hierarchical locations of execution with explicit resource provisioning and to the best of our knowledge the first process algebra capturing notions of virtual time and resource provisioning for nested virtualization. In the proposed model resource provisioning is based on virtual time, a local notion of time reminiscent of time slices for virtual machines in the context of nested virtualization. This way, the computing power of a virtually timed ambient depends on its location in the deployment hierarchy.

To reason about timed behavior in this setting, we define weak timed bisimulation for virtually timed ambients as a conservative extension of bisimulation for mobile ambients, and show that the equivalence of bisimulation and reduction barbed congruence is preserved by this extension.

To perform static analysis of systems of virtually timed ambients, we introduce an *assumption commitment type system* in Chapter 7 for resource management in the context of nested virtualization, which uses assumptions about the outside of a virtually timed ambient to guarantee resource provisioning on the inside. This assumption and commitment type system enables static checking of timing and resource constraints for ambients and gives an upper bound on the resources used by a process. We further prove the soundness of resource consumption for the type system for virtually timed ambients in terms of a subject reduction result, expressing that the upper bounds on resources and on the number of subambients are preserved under reduction, and a progress result, expressing that a well-typed process will not run out of resources.

Furthermore, we combine modal logic for mobile ambients with notions of metric temporal logic in order to define a *modal logic* for virtually timed ambients in Chap-

ter 8, enabling us to describe timed behavior and resource consumption of a system as modal logic properties of processes. We show that the resulting logic is a conservative extension of the modal logic for the ambient calculus, preserving satisfiability. To prove that a process satisfies a formula we provide a model checking algorithm and implement it in the Maude rewriting logic system. Considering modal propositions as modal contracts, we can now prove whether a system satisfies a certain quality of service agreement.

We introduce *resource-awareness* for virtually timed ambients in Chapter 9, which enables dynamic horizontal scaling. We further define weak timed context bisimulation for resource-aware virtually timed ambients as an extension of bisimulation for mobile ambients. We show the feasibility of virtually timed ambients as a modeling language for cloud computing with a case study of dynamic auto scaling on Amazon EC2 modeled in a prototype implementation of our calculus. Lastly, we provide a cloud library for *model implementations* in the rewriting logic system Maude in Chapter 10.

5.2 Discussion of the Research Questions

The stated purpose of this thesis is to formalize virtual environments to enable analysis of cloud computing. We approach this goal by developing a calculus for virtualization, which does not only contain notions of nested locations, mobility and local time but also allows for reasoning about required resources. In order to define the purpose of this thesis more precisely, we stated four research questions, which each, in turn, have been addressed in at least one of the presented research papers.

1. *How can we formalize resource management between distributed locations and allow comparison of different systems?*

We introduce the calculus of virtually timed ambients in Paper 1 by adding a new notion of time and resources to the ambient calculus, which is well-known as a calculus of mobility and nested locations. To compare different systems we use bisimulation, a typical equivalence relation over this kind of calculus. A notion of bisimulation is introduced and thoroughly discussed. Essential properties such as speed and resource consumption are established and investigated.

2. *How can we statically predict the behavior of a system to avoid runtime errors?*

Type systems are a common technique in the field of static analysis, which are used to describe the important features of a calculus and provide a way to have the implementation of those features statically checked. We introduce an assumption commitment type system for resource requirements and provisioning in Paper 2 and prove a subject reduction and a progress result, which are core properties for type systems.

3. *How can we define specifications for resource management as given e.g., in service-level agreements, and allow automatic verification?*

We define modal logic for the calculus of virtually timed ambients in Paper 3. This logic supports the definition of quality of service statements, for example about resource consumption or the nesting structure of a system during the timed reduction of its processes, in the form of logical properties. We further provide a corresponding model checking algorithm and its prototype implementation in Maude, to verify algorithmically that a process satisfies a formula.

4. *How can we enable systems in the defined formalism to react dynamically to changes during runtime and, consequently, to perform dynamic self-management?*

Based on the results of the previous paper we study resource-awareness in Paper 4, allowing the calculus to dynamically react to changes during runtime. We implemented the resulting calculus in the Maude system for rewriting logic, enabling us to define a case study. Furthermore, we provide a cloud library in Maude for easy implementation of different cloud models in Paper 5.

5.3 Outlook on Future Work

Virtualization opens for new and interesting foundational models of computation by explicitly emphasizing deployment and resource management. The research presented in this thesis allows for further investigations in different directions of the topic. The calculus of virtually timed ambients which is developed in this thesis extends the basic ambient calculus with the standard movement capabilities, but an addition of channel communication is not considered in this work. Introducing channels would lead to additional synchronization, which may be used to derive more precise estimations about resource consumption. Such an extension would be non-trivial, as it would involve an analysis of the communication structure and would complicate scheduling.

A natural continuation of the presented work would be the generalization of the scheduling strategy to model different scheduling scenarios. This modification of the calculus would entail a need for the generalization of the type system. Another remaining challenge is the extension of the calculus to model active resource management. This would enable the creation of optimization strategies for resource-aware scaling and entail the development of deployment strategies for asynchronous cloud services. The prototype implementation of the calculus in the Maude system for rewriting logic can be enhanced further with the use of advanced metaprogramming.

Another important aspect of cloud computing that has not been addressed explicitly in this work is the issue of privacy and security. Not only can cloud providers often simply access the data stored on their machines, but also the storage of data from different costumers on the same machine imposes a security risk. As the calculus of mobile ambients was developed with the property to model firewalls in mind, it can be used to formalize authorization and authentication. This makes the addition of security policies to detect and prevent unauthorized intrusion a suitable extension of the calculus of virtually timed ambients.

Part II

Research Papers

CHAPTER 6

Virtually Timed Ambients: A Calculus of Nested Virtualization

Abstract. Nested virtualization enables a virtual machine, which is a software layer representing an execution environment, to be placed inside another virtual machine. Nested virtual machines form a location hierarchy where virtual machines at every level in the hierarchy compete with other processes at that level for processing time. With nested virtualization, the computing power of a virtual machine depends on its position in this hierarchy and may change if the virtual machine moves. This paper introduces the calculus of virtually timed ambients, a formal model of hierarchical locations for execution with explicit resource provisioning, motivated by these effects of nested virtualization. Resource provisioning in this model is based on virtual time slices as a local resource. To reason about timed behavior in this setting, weak timed bisimulation for virtually timed ambients is defined as an extension of bisimulation for mobile ambients. We show that the equivalence of contextual bisimulation and reduction barbed congruence is preserved by weak timed bisimulation. Simulation with time relaxation is defined to express that a system is slower than another system up to a given time bound. The calculus of virtually timed ambients is illustrated by examples.

6.1 Introduction

Virtualization technology enables the resources of an execution environment to be represented as a software layer, a so-called *virtual machine*. Application-level processes are agnostic to whether they run on such a virtual machine or directly on physical hardware. Since a virtual machine is a process, it can be executed on another virtual machine. Technologies such as VirtualBox, VMWare ESXi, Ravello HVX, Microsoft Hyper-V, and the open-source Xen hypervisor increasingly support running virtual machines inside each other in this way. This *nested virtualization*, originally introduced by Goldberg [73], is necessary to host virtual machines with operating systems which

themselves support virtualization [19], such as Microsoft Windows 7 and Linux KVM. Nested virtualization has many uses, for example for end-user virtualization for guests, in development, and in deployment testing. Nested virtualization is also a crucial technology to support the hybrid cloud, as it enables virtual machines to migrate between different cloud providers [159].

To study the logical behavior of virtual machines in the context of nested virtualization, this paper develops a calculus of virtually timed ambients with explicit resource provisioning. Previous work on process algebra with resources typically focusses on binary resources such as locks (e.g., [100, 119]) and previous work on process algebra with time mainly considers timeouts (e.g., [15, 20, 77, 118, 122]). In contrast, time and resources in virtually timed ambients are *quantitative* notions: a process which gets *more resources* typically executes *faster*. Virtually timed ambients can be understood as locations for the deployment of processes; the resource requirements of processes executing at a location are matched by resources made available by the virtually timed ambient. The amount of resources made available by a virtually timed ambient constitutes its computing power. This amount is determined by the time slices the virtually timed ambient receives from its parent ambient. A virtually timed ambient that shares the time slices of its parent ambient with another process has less available time slices to execute its own processes.

The time model used to realize this kind of resource provisioning for virtually timed ambients is here called *virtual time*. Virtual time is provided to a virtually timed ambient by its parent ambient, similar to the time slices that an operating system provisions to its processes. When we consider many levels of nested virtualization, virtual time becomes a *local* notion of time which depends on a virtually timed ambient's position in the location hierarchy. 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 executed in the virtually timed ambient which moves, in the virtually timed ambients it leaves, and in the virtually timed ambient it enters. The model of resource provisioning in virtually timed ambients is inspired by Real-Time ABS [87], but extended to address nested virtualization in our calculus.

To formalize nested virtualization, notions of location mobility and nesting are essential. The calculus of mobile ambients, originally developed by Cardelli and Gordon [43], captures processes executing at distributed locations in networks such as the Internet. Mobile ambients model both location mobility and nested locations, which makes this calculus well-suited as a starting point for our work. Combining these notions from the ambient calculus with the concepts of virtual time and resource provisioning, the calculus of virtually timed ambients can be seen as a model of nested virtualization. To capture migration, virtually timed ambients will have capabilities reminiscent of those for mobile ambients, but the capabilities of virtually timed ambients need to deal with virtual time and the corresponding changes to the resource provisioning. Thus different locations, barriers between locations, barrier crossing, and their relation to virtual time and resource provisioning are important for the virtually timed ambients; the number and position of virtually timed ambients available for processing tasks influences the overall processing time of a program. This allows the

effects of, e.g., load balancing and scaling to be observed using weak timed bisimulation.

Contributions To study the effects of nested virtualization, the main contributions of this paper can be summarized as follows:

- we define a calculus of *virtually timed ambients*, to the best of our knowledge the first process algebra capturing notions of virtual time and resource provisioning for nested virtualization;
- we define *weak timed bisimulation* for virtually timed ambients, and show that weak timed bisimulation is equivalent to reduction barbed congruence [110] with time;
- we define *time relaxation* for virtually timed ambients as a simulation relation allowing deviation by a bounded amount of time.

A short version of this paper appeared in the proceedings of WADT 2016 [88].

6.2 Preliminaries on Mobile Ambients

Mobile ambients [43] have originally been introduced to represent "administrative domains" for processes. The syntax, as well as the semantics we consider, is based on [110] and largely unchanged compared to [43]. The main difference compared to [43] lies in the separation of processes into two levels, as *processes* and *systems*. This distinction is used to simplify proofs in the bisimulation section. Systems characterize the outermost layer of an ambient structure.

The syntax in Table 6.1 represents the basic calculus for mobile ambients. 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. In the calculus, administrative domains for processes, called *ambients*, are represented by names. A process P located in an ambient named m is written $m[P]$.

Ambients can be nested, and the nesting structure can change dynamically. A change of the nesting structure is specified by prefixing a process with a *capability*. There are three basic capabilities. The input capability *in* n indicates the willingness of a process (respectively its containing ambient) to enter an ambient named n, running in parallel with its own ambient; e.g., $k[in\ n.P] \mid n[Q] \rightarrow n[k[P] \mid Q]$. The output capability *out* n enables an ambient to leave its surrounding (or parental) ambient n; e.g., $n[k[out\ n.P] \mid Q] \rightarrow k[P] \mid n[Q]$. The open capability *open* n allows an ambient named n at the same level as the capability to be opened; e.g., $k[open\ n.P \mid n[Q]] \rightarrow k[P \mid Q]$.

	n	name
Systems:		
$M, N ::=$	$\mathbf{0}$	inactive system
	$M \mid N$	parallel composition
	$(\nu n)M$	restriction
	$n[P]$	ambient
Processes:		
$P, Q ::=$	$\mathbf{0}$	inactive process
	$P \mid Q$	parallel composition
	$(\nu n)P$	restriction
	$!C.P$	replication
	$C.P$	prefixing
	$n[P]$	ambient
Capabilities:		
$C ::=$	$\mathit{in}\ n$	can enter n
	$\mathit{out}\ n$	can exit n
	$\mathit{open}\ n$	can open n

Table 6.1: Syntax of the mobile ambient calculus.

6.2.1 Syntax

We use the following notational conventions. Parallel composition has the lowest precedence among the operators and the process $E.F.P$ is read as $E.(F.P)$. For names, the ν-operator acts as a binder and the sets of free names fn of a process is defined as expected. We use $\widetilde{m}$ to denote a tuple of names $m_1, m_2, \ldots, m_k$. Note that Table 6.1, following [110], only allows replicated prefixing $!C.P$. Since replicated input in the π-calculus has the same expressive power as full replication [80] and recursion [115, 146], the replication of an ambient in the ambient calculus can be implemented with replicated prefixing and stuttering. We shall use the notion $!n[]$ in the examples as an abbreviation for this stuttering bypass.

$P \rightarrowtriangle Q \Rightarrow (\nu n)P \rightarrowtriangle (\nu n)Q$	(R-Res)
$P \rightarrowtriangle Q \Rightarrow P \mid R \rightarrowtriangle Q \mid R$	(R-Par)
$P \rightarrowtriangle Q \Rightarrow n[P] \rightarrowtriangle n[Q]$	(R-Amb)
$P' \equiv P, P \rightarrowtriangle Q, Q \equiv Q' \Rightarrow P' \rightarrowtriangle Q'$	(R-Red $\equiv$)
$n[\mathit{in}\ m.P \mid Q] \mid m[R] \rightarrowtriangle m[n[P \mid Q] \mid R]$	(R-In)
$m[n[\mathit{out}\ m.P \mid Q] \mid R] \rightarrowtriangle n[P \mid Q] \mid m[R]$	(R-Out)
$\mathit{open}\ n.P \mid n[Q] \rightarrowtriangle P \mid Q$	(R-Open)

Table 6.2: Reduction rules.

$P \equiv P$	(S-Refl)
$P \mid \mathbf{0} \equiv P$	(S-ZeroPar)
$P \equiv Q \Rightarrow Q \equiv P$	(S-Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(S-Trans)
$P \mid Q \equiv Q \mid P$	(S-ParComm)
$(P \mid Q) \mid R \equiv Q \mid (P \mid R)$	(S-Par-Assoc)
$!C.P \equiv C.P \mid !C.P$	(S-Repl Par)
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(S-ResRes)
$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$ if $n \notin fn(P)$	(S-Res-Par)
$(\nu n)(m[P]) \equiv m[(\nu n)P]$ if $n \neq m$	(S-Res-Amb)

Table 6.3: Structural congruence.

6.2.2 Semantics

The semantics is given as a reduction semantics which combines structural congruence with reduction rules. The reduction relation $P \rightarrowtriangle Q$, which describes the evolution of a process, is captured by the reduction rules in Table 6.2. Structural congruence $P \equiv Q$ relates different syntactic representations of the same process and can be seen in Table 6.3.

6.3 Virtually Timed Ambients

Mobile ambients [43] are located processes, arranged in a hierarchy which may change dynamically. Interpreting the location as a place 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, but on the other hand on the placement or deployment of the component in the hierarchical ambient structure. Virtually timed ambients combine timed processes and timed capabilities with the features of the calculus for mobile ambients summarized above.

6.3.1 Syntax and Semantics

Timed systems and processes are defined analogously to Table 6.1, with the difference that each virtually timed ambient contains a *local clock* and other virtually timed ambients or processes.

Definition 1 (Virtually timed ambients)**.** *Virtually timed ambients are given by the syntax in Table 6.4.*

In Table 6.4 we can see that in the calculus of virtually timed ambients every closed system of ambients must be contained in a root ambient with a *source clock* triggering the clocks of the local subambients recursively. The timed capabilities of virtually timed ambients extend the capabilities of mobile ambients with an additional effect on

	n	name
	`tick`	virtual time slice
Global systems:		
$G ::=$	$\mathbf{0}$	inactive system
	$G \mid G$	parallel composition
	$n[\textsc{Clock}^{\dagger} \mid M]$	virtually timed root ambient
Timed systems:		
$M, N ::=$	$\mathbf{0}$	inactive system
	$M \mid N$	parallel composition
	$(\nu n)M$	restriction
	$n[\textsc{Clock} \mid P]$	virtually timed ambient
Timed processes:		
$P, Q ::=$	$\mathbf{0}$	inactive process
	$P \mid Q$	parallel composition
	$(\nu n)P$	restriction
	$!C.P$	replication
	$C.P$	prefixing
	$n[\textsc{Clock} \mid P]$	virtually timed ambient
Timed capabilities:		
$C ::=$	$\mathbf{in}\ n$	can enter n and adjust the local clock there
	$\mathbf{out}\ n$	can exit n and adjust the local clock on the outside
	$\mathbf{open}\ n$	can open n and adjust own local clock
	$\mathbf{c}$	consumes one resource

Table 6.4: Syntax of the virtually timed ambient calculus.

time management, explained below. In order to define computing power a capacity **c** for resource consumption of processes is added.

The semantics is given as a reduction semantics similar to the semantics of mobile ambients. The rules for structural congruence $P \equiv Q$ are equivalent to the ones for mobile ambients in Table 6.3. The reduction relation $P \rightarrowtriangle Q$ is captured by the rules in Table 6.5 and Table 6.6. In Table 6.5 we make use of the notion of *observables* aka barbs. This well-known concept, originally introduced for the π-calculus [116], captures a notion of immediate observability. In the ambient calculus, what is immediately observable is the presence of a top-level ambient whose name is not restricted. The obervability predicate $\downarrow_n$ or "barb" is defined as follows.

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

By moving the ν-binders to the outside and only taking the inside of their scope into consideration we can observe the bound ambients inside the scope of the ν-binders.

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

6.3.2 Virtual Time and Local Clocks

To represent the outlined time model the local clock contained in each virtually timed ambient is responsible for triggering timed behavior and local resource consumption. Each time slice emitted by a local clock triggers the clock of one of its subambients in a round-robin way or is consumed by a process as a resource. This corresponds to a simple form of fair, *preemptive scheduling*, which makes the system's behavior sensitive to the number of co-located virtually timed ambients and resource consuming processes.

Clocks have a *speed*, interpreted *relative* to the speed of the surrounding virtually timed ambient. The speed of a clock is given by the pair (p, q), where p is the number of local time slices emitted for a number q of time slices received from the surrounding ambient, $p, q \in \mathbb{N}$. Thus, time in a nested ambient is relative to the global time, and depends on the speed of the clocks of the ambients in which it is contained and on its number of siblings.

The speed of the source clocks is defined as a pair $(n, 0)$, where $n \in \mathbb{N}$, as the sources do not need any input, while for the speed of a local clock it holds that an input of $q = 0$ is only valid if $p = 0$, too. As virtually timed ambients with speed $(0, 0)$ do not require any times slices from their parental ambient and do not exhibit any timed behavior, they are not considered *time consuming*. However, processes which are prefixed with the resource consumption capability $\mathbf{c}\,.P$ are considered time consuming. Note that mobile ambients can be represented as virtually timed ambients with a clock with speed $(0, 0)$.

Definition 4 (Local clocks)**.** *A local clock contains a* counter *to record the number of received time slices, its own* speed*, and two* sets*:*

$$\textsc{Clock}\{\mathit{counter}, (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}\ .$$

The first set contains the names of time consuming processes running in the ambient as well as time consuming virtually timed subambients in the surrounding ambient which have not yet received a time slice in the current cycle and the second set those which have.

When a clock receives a time slice, denoted `tick`, from its surrounding ambient, one of the following actions occurs: If $\mathit{counter}+1 < q$, then the clock records this time slice and continues waiting (i.e., $\textsc{Clock}\{\mathit{counter} \coloneqq \mathit{counter}+1, (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}$); if $\mathit{counter}+1 = q$ the input number is reached, the counter is set to 0, the clock emits time slices to p subambients of the first set and puts them in the second

$$\frac{\begin{array}{l}\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \{m, \boxed{n}\}, \emptyset\} \\ \textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \\ \textsc{Clock}_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \ldots, b_i\}, \{b_{i+1}, \ldots b_j\}\} \\ \textsc{Clock}^*_k = \textsc{Clock}\{c_k, (p_k, q_k), \{m\}, \emptyset\} \\ \textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, \ldots, a_k, \boxed{n}\}, \{a_{k+1}, \ldots a_n\}\} \\ \textsc{Clock}^*_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \ldots, b_i\} \cup \boxed{P_\downarrow}, \{b_{i+1}, \ldots b_j\}\}\end{array}}{\begin{array}{l}k[\textsc{Clock}_k \mid n[\textsc{Clock}_n \mid \boxed{\textbf{in}\ m.P} \mid Q] \mid m[\textsc{Clock}_m \mid R]] \\ \quad \twoheadrightarrow k[\textsc{Clock}^*_k \mid m[\textsc{Clock}^*_m \mid R \mid n[\textsc{Clock}_n \mid P \mid Q]]]\end{array}}\ (\text{TR-In})$$

$$\frac{\begin{array}{l}\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \emptyset, \{m\}\} \\ \textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \ldots, a_k, \boxed{n}\}, \{a_{k+1}, \ldots a_n\}\} \\ \textsc{Clock}_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \ldots, b_i\}, \{b_{i+1}, \ldots b_j\}\} \\ \textsc{Clock}^*_k = \textsc{Clock}\{c_k, (p_k, q_k), \{\boxed{n}\}, \{m\}\} \\ \textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \\ \textsc{Clock}^*_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \ldots, b_i\} \cup \boxed{P_\downarrow}, \{b_{i+1}, \ldots b_j\}\}\end{array}}{\begin{array}{l}k[\textsc{Clock}_k \mid m[\textsc{Clock}_m \mid n[\textsc{Clock}_n \mid \boxed{\textbf{out}\ m.P} \mid Q] \mid R]] \\ \quad \twoheadrightarrow k[\textsc{Clock}^*_k \mid n[\textsc{Clock}_n \mid P \mid Q] \mid m[\textsc{Clock}^*_m \mid R]]\end{array}}\ (\text{TR-Out})$$

$$\frac{\begin{array}{l}\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{\boxed{n}\}, \emptyset\} \\ \textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{\boxed{a_1, a_2, \ldots, a_n}\} \cup \boxed{P_\downarrow}, \emptyset\}\end{array}}{m[\textsc{Clock}_m \mid \boxed{\textbf{open}\ n.P} \mid n[\textsc{Clock}_n \mid R] \mid Q] \twoheadrightarrow m[\textsc{Clock}^*_m \mid P \mid R \mid Q]}\ (\text{TR-Open})$$

$$\frac{\begin{array}{l}\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \ldots, a_n\}, \emptyset\} \\ \textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \ldots, a_n, \boxed{\mathbf{c}\ .P}\}, \emptyset\} \\ \boxed{p_m > 0}\end{array}}{m[\textsc{Clock}_m \mid \boxed{\mathbf{c}\ .P} \mid R] \twoheadrightarrow m[\textsc{Clock}^*_m \mid R]}\ (\text{TR-Resource})$$

Table 6.5: Timed reduction rules for timed capabilities, where $a_1, a_2, \ldots, a_n$ are time consuming virtually timed ambients and processes in R and $b_1, b_2, \ldots, b_j$ in Q, respectively. Here a blue backdrop marks the trigger of the reduction, red the changes in the clocks and green eventual constraints.

$$\frac{\textsc{Clock} = \textsc{Clock}\{\, c \,, (p,q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \quad \textsc{Clock}^* = \textsc{Clock}\{\, c+1 \,, (p,q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \quad c+1 < q}{m[\,\texttt{tick} \mid \textsc{Clock} \mid R] \twoheadrightarrow m[\textsc{Clock}^* \mid R]} \text{ (TR-Tick}_1\text{)}$$

$$\frac{\textsc{Clock} = \textsc{Clock}\{c, (p,q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \quad c+1 = q}{m[\,\texttt{tick} \mid \textsc{Clock} \mid R] \twoheadrightarrow m[\,\mathrm{RR}(\textsc{Clock} \mid R)\,]} \text{ (TR-Tick}_2\text{)}$$

$$\frac{\textsc{Clock}^\dagger = \textsc{Clock}^\dagger\{0, (n,0), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}}{\textsc{Clock}^\dagger \mid R \twoheadrightarrow \mathrm{RR}(\textsc{Clock}^\dagger \mid R)} \text{ (TR-Source)}$$

Table 6.6: Timed reduction rules, where $a_1, a_2, \ldots, a_n$ are time consuming virtually timed ambients and processes in R, and for (TR-Source) on the top level. Here a blue backdrop marks the trigger of the reduction, red the changes in the clocks and green eventual constraints.

set (i.e., $\textsc{Clock}\{\mathit{counter} \coloneqq 0, (p,q), \{a_{p+1} \ldots, a_k\}, \{a_{k+1}, \ldots a_n, a_1, a_2, \ldots, a_p\}\}$). As soon as the first of the two sets is empty, the first and second set are switched. Thus, no ambient receives a second time slice before every other subambient has received the first one. In the following, we omit the representation of the counter and the sets of subambients. For a better overview in the examples we denote the speed of the clocks as superscript $\textsc{Clock}^{p,q}$. If an ambient is not time consuming, i.e, it has a clock with speed $(0,0)$, we do not mention the clock. For actions which do not require time we assume *maximal progress*. In terms of structural congruence a clock is treated analogously to a process. The following example shows the encoding of a system with a load balancer in virtually timed ambients.

Example 1 (System with load balancer)**.** *A system with a load balancer can be defined as follows:*

$$\begin{aligned}
&\textit{load balancer system: } (\nu\ lb, a, b)\ lbs[\textsc{Clock}^{2,1} \mid lb[\ldots] \mid a[\ldots] \mid b[\ldots]] \\
&\quad \textit{incoming request: } request[P.done_signal \mid \textit{in } lbs.enter_signal.\ \textbf{open}\ move] \\
&\quad\quad \textit{load balancer: } lb[!\ \textbf{open}\ start.wait_for_enter.\ \textbf{open}\ lock_a. \\
&\qquad\qquad wait_for_enter.\ \textbf{open}\ lock_b.start[] \mid !lock_a[x[]] \mid \\
&\qquad\qquad !lock_b[y[]] \mid !(\textbf{open}\ x.move[\textbf{out}\ lb.\ \textbf{in}\ request.\ \textbf{in}\ a] \mid \\
&\qquad\qquad\quad \textbf{open}\ y.move[\textbf{out}\ lb.\ \textbf{in}\ request.\ \textbf{in}\ b])] \\
&\quad\quad\quad \textit{ambient a: } a[\textsc{Clock}^{1,1} \mid !\ \textbf{open}\ request.wait_for_done. \\
&\qquad\qquad done[\textbf{out}\ a.\ \textbf{out}\ lbs]] \\
&\quad\quad\quad \textit{ambient b: } b[\textsc{Clock}^{1,1} \mid !\ \textbf{open}\ request.wait_for_done.
\end{aligned}$$

$$done[\textbf{out}\ b.\ \textbf{out}\ lbs]].$$

Here, the untimed load balancer creates a move ambient which moves incoming requests alternately into the virtually timed ambients a and b. For each time slice it receives from the source clock of the surrounding root ambient, the local clock of lbs distributes two time slices. Therefore, both subambients a and b receive one time slice. When a request has been executed, it releases an ambient done which emerges to the outside of the system and becomes observable. The enter- and done-signals are shorthand notions, but can easily be implemented in the calculus of virtually timed ambients.

RR: Round-based scheduling function

input: CLOCK$\{counter, (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots, a_n\}\} \mid R$
$S := \{a_1, a_2, \ldots, a_k\}$
$T := \{a_{k+1}, \ldots, a_n\}$
if $S = \emptyset$ **then**
return CLOCK$\{0, (p, q), \emptyset, \emptyset\}$
else
  while $p \geq |S|$ **do**
    for all $a_i \in S$ **do**
      if $a_i =$**c** $.P$ **then** $S := S \setminus a_i$; $R := R \mid P$
        $S := S \cup P_{\downarrow}$
      else $R := a_1 \mid \cdots \mid a_i[\texttt{tick} \mid \cdots] \mid \cdots \mid a_n$
      end if
    end for
    $p := p - |S|$; $S := S \cup T$; $T := \emptyset$
  end while
  Choose a subset $S' \subset S$ such that $|S'| = p$.
  for all $a_i \in S'$ **do**
    if $a_i =$**c** $.P$ **then** $S' := S \setminus a_i$, $R := R \mid P$
      $S := S \cup P_{\downarrow}$
    else $R := a_1 \mid \cdots \mid a_i[\texttt{tick} \mid \cdots] \mid \cdots \mid a_n$
    end if
  end for
  $S := S \setminus S'$; $T := T \cup S'$
end if
return CLOCK$\{0, (p, q), S, T\} \mid R$

6.3.3 Timed Capabilities

The timed capabilities **in** n, **out** n, and **open** n enable virtually timed ambients to move in a timed system. When moving virtually timed ambients, we must consider

that the clocks need to know about their current subambients, therefore their list of subambients need to be adjusted.

We now explain the reduction rules for virtually timed ambients, which are given in Table 6.5 and Table 6.6. Observe that if we would not adjust the clocks then the moving subambient would not receive time slices from its new parental clock. In (TR-In) and (TR-Out), the clocks of the old and new parental ambient of the moving ambient have to be updated. In (TR-Open) the clock of the opening ambient itself is updated. Note also that here the clock of the opened ambient is deleted. For virtually timed ambients with a clock with speed $(0, 0)$, the timed capabilities are equivalent to the capabilities for mobile ambients, as ambients, which are not time consuming, are not considered in the time management of the clocks. In (TR-Resource) the time consuming process is moved into the clock, 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 $p > 0$, meaning ambients which actually emit resources. Ambients which do not emit resources can therefore be used to safely transfer request between ambients which are used as computation environments without interfering with the contents of the requests. In (TR-Tick$_1$) the required number q of input time slices to trigger the local clock is not reached, thus the incoming time slice, denoted as `tick`, is only registered in the counter. In (TR-Tick$_2$) the local clock releases p time slices to its subambients and potentially to time consuming processes. This is denoted with the function RR for round-based scheduling of time slices. The function takes as input the Clock and distributes time slices `tick` to the subambients and processes in the given sets, thereby adjusting the sets of remaining and of served ambients. The source clocks Clock† can reduce without parental time slices as given in (TR-Source).

6.3.4 Resource Consumption

Processes expend the processing power of the ambient they are contained in by consuming the local time slices as resources. Thus, time consuming processes and time consuming subambients in a virtually timed ambient compete for the same resource. The consumption of a computing resource is defined as the capability **c**. A process P without any appearance of **c** is called *not time consuming*. An ambient with a higher local clock speed produces more time slices and therefore also more resources for each parental time slice, which in turn allows more work to be done for each parental time slice.

We illustrate resource consumption by considering a request which was sent to the system of Example 1.

Example 2 (Resource consumption)**.** *Consider the virtually timed system with a load balancer from Example 1, with an incoming request.*

$$lbs[\cdots] \mid request[\mathbf{c}\ .\ \mathbf{c}\ .done\_signal \mid \mathbf{in}\ lbs.enter\_signal.\ \mathbf{open}\ move]$$

The request enters the system and is transferred by the load balancer into a, where it is opened during the reduction and awaits resource consumption. After one time signal of the source clock, the virtually timed ambient a emits one resource, which is consumed by the request:

$$a[\text{CLOCK}^{1,1} \mid !open\ request.wait\_for\_done.done[\textbf{out}\ a.\ \textbf{out}\ lbs] \\ \mid wait\_for\_done.done[\textbf{out}\ a.\ \textbf{out}\ lbs] \mid \textbf{c}\ .done\_signal].$$

After another time signal from the source clock the ambient with name done can emerge to the top level:

$$a[\text{CLOCK}^{1,1} \mid !open\ request.wait\_for\_done.done[\textbf{out}\ a.out\ lbs]] \mid \\ done[\textbf{out}\ lbs].$$

6.3.5 Accumulated Speed

The *accumulated speed* (a_m, b_m) in an ambient m is the relative speed of the ambient with respect to the source clock and the siblings. As the clocks distribute time slices in a form of preemptive scheduling, such that each child gets one time slice in a round robin way, it holds that the accumulated speed of an ambient is influenced by the parental speed and the number of children n in the parental ambient. Thus, this approach is not only *path sensitive*, but also *sibling sensitive*.

Definition 5. *Let (a_{parent}, b_{parent}) be the accumulated speed in the direct parental ambient of an ambient m, $n_{children}$ the number of children of the parent, and C the chain of all parental ambients of m up to the global level, then the accumulated speed for preemptive scheduling in a subambient m is given as follows:*

$$\begin{aligned}(a_m, b_m) &= \left(p_m, \left\lceil q_m \cdot n_{children} \cdot \frac{max\{1, b_{parent}\}}{a_{parent}} \right\rceil\right) \\ &= \left(p_m, \left\lceil q_m \cdot \prod_{k \in C} n_{children\ of\ k} \cdot \prod_{k \in C} \frac{max\{1, b_k\}}{a_k} \right\rceil\right)\end{aligned}$$

Example 3 (Change of accumulated speed)**.** *Consider the virtually timed system lbs defined in Example 1, we define now a new system lbs_c which equals lbs, except that it contains a third subambient c.*

$$\begin{aligned}\textit{system: } &(\nu\ lb, a, b, c)\ lbs_c[\text{CLOCK}^{2,1} \mid lb \mid a \mid b \mid c] \\ \textit{ambient c: } &c[\text{CLOCK}^{1,2} \mid !\ \textbf{open}\ request.wait\_for\_done.done[\textbf{out}\ c.out\ lbs_c]]\end{aligned}$$

As the approach is sibling sensitive, the accumulated speeds of the subambients of the system are reduced compared to the setup in Example 1 with only two subambients. Before, the accumulated speed of both a and b was $(1, 1)$, now the speed of a and b is $(1, 2)$ and the accumulated speed of c is $(1, 3)$.

6.4 Bisimulation and Barbs for Virtually Timed Ambients

When comparing the behavior of virtually timed ambients, we want to consider time as a factor. We first introduce a timed version of bisimulation and later of reduction barbed congruence.

The concept of (bi)simulation is an important, well-established, and extensively studied technique to define equivalences of concurrent or reactive systems [145]. It comes in many flavors and may or may not be a congruence, as well, depending on the constructs of the language and minutiae of the definition and the semantics. Being a congruence, of course, is generally intended, as that allows compositional arguments about equivalence of systems and replacing a subsystem by "equivalent" ones, without changing the overall behavior. A very important sub-class of bisimulations are so-called *weak* bisimulations; they are based on a distinction between observable and non-observable or internal actions, ignoring the latter. Ignoring internal behavior as unobservable is essential for being a useful basis of comparison, but unfortunately the issue of being a congruence or not becomes more tricky.

This section will define a notion of weak bisimulation for virtually timed ambients, generalizing the formalization for mobile ambients from [110], taking care of the timed behavior. To relate systems concerning their timed behavior or speed, in particular the "ticks" of the global clocks will be treated as observable. The standard notion bisimulation (weak or strong) relates two systems via the *transitions* each system makes, requiring that each (observable or every) step one system takes is mimicked accordingly by the other system, and vice versa.

In that setting, the steps of a system represent its atomic interactions with its environment, modeled as *labelled* transitions. The *reduction* semantics from Section 6.3, however, captures the behavior of *closed* or global systems. The semantics is formalized as (unlabelled) *reduction* steps $\rightarrowtriangle$ (additionally assisted by structural congruence rules), i.e., without interaction with the environment or with a surrounding context. To define bisimulation, we first formalize an *open* version of the operational semantics, using a labelled transition relation. To express interaction with a surrounding *context* in the open setting, transitions will have *labels* which capture interaction with an environment.

For this purpose, we define a semantics of virtually timed ambients based on labeled transition systems, which can perform global time steps as observable actions. The semantics is intended, of course, not as a semantically different alternative to the reduction semantics, but as capturing the same behavior, described from the perspective of open systems. For that transition semantics, we define a notion of weak timed bisimulation (cf. Section 6.4.1). Section 6.4.2 establishes that the introduced timed notion of bisimulation is a congruence. Following a standard line of development, this is done indirectly. First, one need to define a notion of bisimulation for the original semantics. It's defined contextually and based on so-called barbs, thereby fitting the closed-system reduction semantics. The resulting definition is a congruence, and the

second part of the argument establishes that both definitions of bisimulation coincide.

6.4.1 Weak Bisimulation for Virtually Timed Ambients

Let us start by defining the available labels for the transition system semantics.

Definition 6 (Labels)**.** *Let the set of labels Lab, with typical element α, be given as follows:*

$$\begin{array}{rcl} \alpha \in Lab & ::= & \tau \\ & \mid & k.\texttt{enter\_}n \mid k.\texttt{exit\_}n \mid k.\overline{\texttt{enter\_}}n \mid n.\texttt{open\_}k \\ & \mid & *.\texttt{exit\_}n \mid *.\texttt{enter\_}n \\ & \mid & k.\texttt{tick} \end{array}$$

where k and n represent names of ambients. τ is the internal label*, the rest are called* observable *labels. We refer to labels of the forms $*.\texttt{exit\_}n$ and $*.\texttt{enter\_}n$ as* anonymous *and other labels as* non-anonymous*, and let the* untimed *labels exclude the tick labels.*

The behavior of a timed system interacting with its environment is given as a transmission system with transition labels from *Lab*. Note that the **c** capability does not represent an interaction with an environment but an internal action and is therefore not captured by a separate observable label apart from τ.

Definition 7 (Timed labeled transitions)**.** *The observable steps $M \xrightarrow{\alpha} M'$ of the timed labeled transition semantics for timed systems is given by the rules of Table 6.7. For internal behavior, τ-steps are the result of reduction steps, i.e., $M \twoheadrightarrow M'$ implies $M \xrightarrow{\tau} M'$.*

The untimed labels, recording the system-environment interactions, i.e., ambient movements induced by the capabilities, coincide with the ones from the untimed case of mobile ambients [110].

In rules (ENTER) and (EXIT), an ambient k enters, respectively exits, from an ambient n provided by the environment. The rules (ENTER SHH) and (EXIT SHH) model the same behavior for ambients with private names. In rule (CO-ENTER), an ambient n, provided by the environment, enters an ambient k of the process. In rule (OPEN), the environment provides an ambient n in which the ambient k of the process is opened. In rule (TICK), the transition $M \xrightarrow{k.\texttt{tick}} M'$ expresses that the top-level ambient k of the system M receives one time slice `tick` from the source clock on the global level. Note that the post-configurations after the transitions contain the symbol $\circ$, which is used as placeholder variable. The labels, capturing interaction with the environment, carry partial information about the "data" exchanged with the environment. For example, label $k.\texttt{enter\_}n$ carries information about the identity k of the ambient being entered, which is contained in the system, as well as about the identity of the one entering named n, which, before the step, is still part of the environment. If thus the enter-label conceptually indicates that some arbitrary ambient $n[R \mid \text{CLOCK}]$ enters the system as effect of executing the **in** n-capability, then the

$(\nu\widetilde{m})(m[\textsc{Clock} \mid \mathbf{in}\ n.P \mid Q] \mid M), m \in \widetilde{m}$ $\xrightarrow{*.\texttt{enter}\_n} (\nu\widetilde{m})(n[m[(\textsc{Clock} \mid P) \mid Q] \mid \circ\ ] \mid M)$	(Enter Shh)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid \mathbf{in}\ n.P \mid Q] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\texttt{enter}\_n} (\nu\widetilde{m})(n[k[(\textsc{Clock} \mid P) \mid Q] \mid \circ\ ] \mid M)$	(Enter)
$(\nu\widetilde{m})(m[\textsc{Clock} \mid \mathbf{out}\ n.P \mid Q] \mid M), m \in \widetilde{m}$ $\xrightarrow{*.\texttt{exit}\_n} (\nu\widetilde{m})(m[(\textsc{Clock} \mid P) \mid Q] \mid n[M \mid \circ\ ])$	(Exit Shh)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid \mathbf{out}\ n.P \mid Q] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\texttt{exit}\_n} (\nu\widetilde{m})(k[(\textsc{Clock} \mid P) \mid Q] \mid n[M \mid \circ\ ])$	(Exit)
$(\nu\widetilde{m})(k[(\textsc{Clock} \mid P)] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\overline{\texttt{enter}}\_n} (\nu\widetilde{m})(k[\textsc{Clock}^* \mid n[\circ] \mid P] \mid M)$	(Co-Enter)
$(\nu\widetilde{m})(k[(\textsc{Clock} \mid P)] \mid M)$ $\xrightarrow{n.\texttt{open}\_k} n[\circ \mid (\nu\widetilde{m})(P \mid M)]$	(Open)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid Q] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\texttt{tick}} (\nu\widetilde{m})(k[\textsc{Clock} \mid \texttt{tick} \mid Q] \mid M)$	(Tick)

Table 6.7: Rules for timed labeled transition systems, where in (Co-Enter) given $\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \{a_1, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}$ the updated clock is denoted by $\textsc{Clock}_k^* = \textsc{Clock}\{c_k, (p_k, q_k), \{a_1, \ldots, a_k, n\}, \{a_{k+1}, \ldots a_n\}\}$ as seen in Table 6.5.

name n is mentioned as part of the label but its "body" $R \mid$ Clock is not. Later, when relating the respective actions of two systems via a notion of bisimulation, intuitively, if one system does a transition where $n[R \mid \textsc{Clock}]$ enters, the second system must be able to exhibit the same transition, i.e., have the "same" ambient entering without breaking their (bi)simulation relationship. In principle, though, the second system can simulate the first doing a step where an ambient $n[R]$ enters, with the body $S \equiv R \mid \textsc{Clock}$. To achieve that (without overburdening the labels by interpreting them up-to structural congruence $\equiv$), the definition will make use of the placeholder $\circ$ and requiring preservation of the relationship for all *instantiations* of the placeholders for both systems by the same body (cf. Definition 8 below). The substitution of the placeholder by a pair of process and it local clock is written as $P \bullet (\textsc{Clock} \mid Q)$ and defined as expected.

The reduction semantics of a process can be encoded in the labelled transition system, because a reduction step can be seen as an interaction with an empty context. We are interested in bisimulations that abstract from τ-actions and use the notion of *weak actions*; let $\Longrightarrow$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$, let $\overset{\alpha}{\Rightarrow}$ denote $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$, and let $\overset{\hat{\alpha}}{\Rightarrow}$ denote $\Longrightarrow$ if $\alpha = \tau$ and $\overset{\alpha}{\Rightarrow}$ otherwise.

An example of a system consuming one parental `tick` and performing the subse-

quent τ-actions is given below:

Example 4 (Timed transition)**.** *Let us reconsider Example 2. After one time signal of the source clock, the virtually timed ambient a emits one resource, which is consumed by the request:*

$$\begin{aligned} a[P'] := \; & a[\textsc{Clock}^{1,1} \mid\!! \; \mathbf{open}\ \mathit{request}.\mathit{wait\_for\_done}.\mathit{done}[\mathbf{out}\ a.\ \mathbf{out}\ \mathit{lbs}] \\ & \mid \mathit{wait\_for\_done}.\mathit{done}[\mathbf{out}\ a.\ \mathbf{out}\ \mathit{lbs}] \mid\!\mathbf{c}\ .\mathit{done\_signal}]. \end{aligned}$$

After another time signal from the source clock and some internal τ steps the ambient named done can emerge, thus here it holds that:

$$\begin{aligned} & \mathit{lbs}[\textsc{Clock}^{2,1} \mid \mathit{lb} \mid a[P'] \mid b] \xrightarrow{\mathit{lbs}.\mathtt{tick}} \mathit{lbs}[\textsc{Clock}^{2,1} \mid \mathtt{tick} \mid \mathit{lb} \mid a[P'] \mid b] \\ & \Longrightarrow \mathit{lbs}[\textsc{Clock}^{2,1} \mid \mathit{lb} \mid a \mid b] \mid \mathit{done}[]. \end{aligned}$$

For virtually timed ambients from Table 6.4 and their labelled transition system, we now define the notion of weak timed bisimulation, where `tick` steps are counted among the observable transitions.

Definition 8 (Weak timed bisimulation)**.** *A symmetric relation $\mathcal{R}$ over timed systems is a* weak timed bisimulation *if $M\ \mathcal{R}\ N$ and $M \xrightarrow{\alpha} M'$ implies:*

1. *If α is a non-anonymous label, then $N \stackrel{\hat{\alpha}}{\Longrightarrow} N'$ for some N', such that*

$$M'\bullet(\textsc{Clock} \mid P) \;\; \mathcal{R} \;\; N'\bullet(\textsc{Clock} \mid P)$$

   (for all clocks Clock *and processes P).*

2. *For anonymous labels:*

   (a) *If $\alpha = *.\mathtt{enter}\_n$, then $N \mid n[\circ] \Longrightarrow N'$ for some N', such that*

$$M'\bullet(\textsc{Clock} \mid P) \;\; \mathcal{R} \;\; N'\bullet(\textsc{Clock} \mid P)$$

   (for all clocks Clock *and processes P).*

   (b) *If $\alpha = *.\mathtt{exit}\_n$, then $n[\ \circ \mid N] \Longrightarrow N'$ for some N', such that*

$$M'\bullet(\textsc{Clock} \mid P) \;\; \mathcal{R} \;\; N'\bullet(\textsc{Clock} \mid P)$$

   (for all clocks Clock *and processes P).*

Systems M and N are *weakly timed bisimilar*, written $M \approx^t N$, if $M\ \mathcal{R}\ N$ for some weak timed bisimulation $\mathcal{R}$. If two systems are weakly timed bisimilar in a timed setting where we observe the ticking of the source clock, then it follows from the definition of weak timed bisimulation that they are weakly bisimilar in a setting where we do not observe the ticking of the clocks but interpret all `tick`-actions as τ-actions, instead. Since strong versions of bisimulation tend to be less useful [146], we mainly consider weak bisimulation and let unqualified terms such as bisimulation and bisimilarity refer to these weak versions.

Lemma 1 (Consistency)**.** *$M \approx^t N$ implies that M and N are weakly bisimilar, $M \approx N$.*

Note that for virtually timed ambients which are not time consuming, i.e. with a speed of $(0, 0)$, weak timed bisimulation and weak bisimulation coincide.

The following example illustrates that systems can be weakly bisimilar in a setting where time is not observed without being weakly timed bisimilar in a timed setting.

Example 5 (Comparing systems with load balancers)**.** *We compare the behavior of the system lbs from Example 1, which we will now call N, with a second system called M, which is defined as follows:*

$$\begin{aligned}
&\textit{load balancer system } M\textit{: } (\nu\ lb, a)\ s[\textsc{Clock}^{1,1} \mid lb \mid a] \\
&\textit{incoming request: } request[P.done_signal \mid \mathbf{in}\ s.enter_signal.\ \mathbf{open}\ move] \\
&\textit{load balancer: } lb[\ !wait_\ for\ _enter.move[\mathbf{out}\ lb.\ \mathbf{in}\ request.\ \mathbf{in}\ a]] \\
&\textit{ambient } a\textit{: } a[\textsc{Clock}^{2,1} \mid !\ \mathbf{open}\ request.wait_\ for\ _done.done[\mathbf{out}\ a.\ \mathbf{out}\ s]]
\end{aligned}$$

In contrast to N, system M only contains one virtually timed ambient, which receives all requests. If we do not observe time, the systems behave the same, as they both answer requests by emitting an observable done-signal. However, the systems are not weakly timed bisimilar:

$$N \approx M \quad \textit{and} \quad N \not\approx^t M\ .$$

We will revisit the example in the next section, after having defined the notion of contexts (cf. Example 6 below).

6.4.2 Reduction Barbed Congruence

The purpose of this section is to establish that weak timed bisimulation $\approx^t$ is a congruence. The result is established indirectly via a second equivalence, reduction barb congruence (cf. Definition 14 below). This relation is defined as the largest relation which is preserved by all constructs of the language (and thereby a congruence by definition), preserved by the internal steps of the reduction semantics, and finally preserved by so called barbs, which are simple observables of terms. Honda and Yoshida's method [81] can be used to define weak reduction barbed congruence for mobile ambients [110]. This approach can be extended to virtually timed ambients.

Definition 9 (Contexts)**.** *A* context *is a process with a hole. A* system context *is a context that transforms systems into systems. System contexts are generated by the following grammar:*

$$\begin{aligned}
\mathcal{C}[-] \ ::= \ & - \ \mid \ \mathcal{C}[-] \mid M \ \mid \ M \mid \mathcal{C}[-] \\
& \mid \ (\nu n)\mathcal{C}[-] \ \mid \ n[\mathcal{C}[-] \mid P] \ \mid \ n[P \mid \mathcal{C}[-]]\ ,
\end{aligned}$$

where M is an arbitrary system and P is an arbitrary process.

Example 6 (Comparing systems with load balancers (2))**.** *Revisiting Example 5, we use the notion of contexts to show that* $N \approx M$ *and* $N \not\approx^t M$*. The argument follows a so-called* up-to proof technique *(e.g., [146]). There are two parts to establish.*

1. *For the untimed equality* $N \approx M$*, we define a relation* $\mathcal{R}$ *as follows:*

$$\mathcal{R} = \{(M_1, M_2) \mid \begin{array}{l} M_1 = (\ (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R]\ , \\ M_2 = (\nu\ lb, a)\ s[lb \mid a \mid R]), \text{for arbitrary } s, R \\ \text{s.t. } lb, a, \text{ and } b \notin fn(R)\ \}\ . \end{array}$$

We need to confirm then, that $\mathcal{R}$ *is a bisimulation up to context and up to structural congruence. So, assume* $N \xrightarrow{\alpha} N'$ *and proceed by case analysis on the structure of* α*.*

Case: $\alpha = \tau$ *(internal action).*
Depending on the nature of N'*, there are three subcases to consider.*

Subcase: $N' \equiv (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R']$*,*
with $R \xrightarrow{\tau} R'$*. It follows that* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{\tau} M'$*, where* $M' \equiv s(\nu\ lb, a)\ [lb \mid a \mid R']$ *and* $N' \equiv \mathcal{R} \equiv M'$

Subcase: $N' \equiv (\nu n)(\nu\ lb, a, b)\ (m[R'] \mid s[lb \mid a \mid b \mid R''])$ *,*
with $R \equiv (\nu n)(m[\mathbf{out}\ s \mid R'] \mid R'')$*. It follows that* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{\tau} M'$*, where* $M' \equiv (\nu n)(m[R'] \mid (\nu\ lb, a)\ s[lb \mid a \mid R''])$*. Now we can factor out the system context* $C[-] = (\nu n)(m[R'] \mid -)$*, thus we are still in* $\mathcal{R}$ *up to context and up to structural congruence.*

Subcase: $N' \equiv (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R] \mid done[]$ *.*
This must have been derived from $(\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R \mid request[P]]$*, for an adequate* P*. This in turn is an instantiation of* $s.\overline{\texttt{enter}}\_request$*. It holds that* $(\nu\ lb, a)\ s[lb \mid a \mid R \mid request[P]]$ *reduces to* $(\nu\ lb, a)\ s[lb \mid a \mid R] \mid done[]$*. Now we can factor out the system context* $C[-] = done[] \mid -$*, thus we are still in* $\mathcal{R}$ *up to context and up to structural congruence.*

Case: $\alpha = s.\texttt{enter}\_n$*.*
Then $N' \equiv n[(\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R'] \mid \circ]$ *and* R *must have execited the capability* **in** n*, reducing to* R'*. Thus,* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{s.\texttt{enter}\_n} M'$*, where* $M' \equiv n[(\nu lb, a)\ s[lb \mid a \mid R'] \mid \circ]$*. Now we can factor out the system context* $C[-] = n[\circ \mid -]$*, thus we are still in* $\mathcal{R}$ *up to context and up to structural congruence.*

Case: $\alpha = s.\texttt{exit}\_n$*.*
Then $N' \equiv (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R] \mid n[\circ]$ *and* R *must have unleashed the capability* **out** n*, reducing to* R'*. Thus,* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{s.\texttt{exit}\_n} M'$*, where* $M' \equiv (\nu\ lb, a)\ s[lb \mid a \mid R'] \mid n[\circ]$*. Now we can factor out the system context* $C[-] = n[\circ] \mid -$*, thus we are still in* $\mathcal{R}$ *up to context and up to structural congruence.*

Case: $\alpha = n.\texttt{open}\_s$.
Then $N' \equiv (\nu\ lb, a, b)\ n[lb \mid a \mid b \mid R \mid \circ]$ *and* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{n.open\_s} M'$, *where* $M' \equiv (\nu\ lb, a)\ n[lb \mid a \mid R \mid \circ]$. *By the definition of* $\mathcal{R}$ *it holds that* $N' \bullet (\textsc{Clock} \mid P) \equiv \mathcal{R} \equiv M' \bullet (\textsc{Clock} \mid P)$ *for all clocks* Clock *and processes* P.

Case: $\alpha = s.\overline{\texttt{enter}}\_k$.
Then $N' \equiv (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R \mid n[\circ]]$ *and* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{s.\overline{\texttt{enter}}\_k} M'$, *where* $M' \equiv (\nu\ lb, a)\ s[lb \mid a \mid R \mid n[\circ]]$. *By the definition of* $\mathcal{R}$ *it holds that* $N' \bullet (\textsc{Clock} \mid P) \equiv \mathcal{R} \equiv M' \bullet (\textsc{Clock} \mid P)$ *for all clocks* Clock *and processes* P.

It follows that $N \approx M$.

2. *For the timed setting we need to show* $N \not\approx^t M$. *Assume for a contradiction that* $N \approx^t M$. *Consider the case* $\alpha = \tau$ *with the subcase* $M' \equiv M'' \mid done[]$. *This must have been derived from* $(\nu\ lb, a)\ s[lb \mid a \mid R \mid P]$, *for an adequate* P. *This in turn is an instantiation of* $s.\overline{\texttt{enter}}\_n$. *Let* $P = request[\mathbf{c}\ .\ \mathbf{c}\ .done\_signal \mid enter\_signal.\ \mathbf{open}\ move]$. *Then it holds that* $(\nu\ lb, a)\ s[lb \mid a \mid R \mid P] \stackrel{s.\texttt{tick}}{\Longrightarrow} (\nu\ lb, a)\ s[lb \mid a \mid R] \mid done[]$. *As* $\approx^t$ *is a weak timed bisimulation it has to hold that* $(\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R \mid P] \stackrel{s.\texttt{tick}}{\Longrightarrow} (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R] \mid done[]$ *for this process* P *as well. However, in the case of system* N *the reduction result* $(\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R] \mid done[]$ *can not be derived with only one execution of* `tick`. *As clocks and resources are restricted to their parental ambients no additional* `tick` *can be added via contexts, and it holds that* $N \stackrel{s.\texttt{tick}}{\Longrightarrow} N'$, *where* $N \equiv (\nu\ lb, a, b)\ s[lb \mid a[P'] \mid b \mid R]$. *Thus,* $M' \approx^t N'$ *does not hold.*

Definition 10 (Preservation under contexts)**.** *A relation* $\mathcal{R}$ *is* preserved by system contexts *if* $M\ \mathcal{R}\ N$ *implies* $\mathcal{C}[M]\ \mathcal{R}\ \mathcal{C}[N]$ *for all system contexts* $\mathcal{C}[-]$.

Theorem 1. *Weak timed bisimilarity is preserved by system contexts.*

Proof. The proof is similar to the related proof for mobile ambients by Merro and Zappa Nardelli [110], and extended for the $k.\texttt{tick}$ action. As $k.\texttt{tick}$ be seen as a special case of the $k.\overline{\texttt{enter}}\_n$ action, the same proof method can be used here. □

We extend the notion of barbs from Definition 2 by a weak version.

Definition 11 (Weak barbs)**.** *Process* P *weakly barbs on* n, $P{\Downarrow_n}$ *if* $P \Longrightarrow P'$ *and* $P'{\downarrow_n}$, *for some* P' .

Besides being preserved by contexts, the congruence definition stipulates preservation under basic observations.

Definition 12 (Preservation of barb)**.** *A relation* $\mathcal{R}$ *over processes is* barb preserving *if* $P\ \ \mathcal{R}\ \ Q$ *and* $P{\downarrow_n}$ *implies* $Q{\Downarrow_n}$.

Definition 13 (Preservation under reduction)**.** *A relation is* reduction closed *(or preserved under reduction) if* $P\ \mathcal{R}\ Q$ *and* $P \rightarrowtriangle P'$, *implies* $Q \rightarrowtriangle Q'$ *for some* Q'

Definition 14 (Reduction barbed congruence over timed systems)**.** Reduction barbed congruence over timed systems $\simeq_s$ *is the largest symmetrical relation over timed systems which is preserved by all system contexts, is reduction closed and barb preserving.*

To show that reduction barbed congruence for timed systems and weak timed bisimilarity coincide, the more challenging direction to establish is the inclusion $\simeq_s \subseteq \approx^t$. Contrapositively formulated it means any two open systems distinguishable by $\approx^t$, are also distinguishable via $\simeq_s$, i.e., via a context which makes this distinction. To do this we need to define a system context that (in particular) observes the action $k.\texttt{tick}$; contexts to observe the other non-anonymous actions of the labeled transition system are defined as for mobile ambients [110]. Again, we can consider $k.\texttt{tick}$ as a special case of the $k.\overline{\texttt{enter}}\_n$ action and can therefore define the tick-observing context as follows:

$$\mathcal{C}_{n.\texttt{tick}}[-] = (\nu a, b)a[\textbf{in}\ n.\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\ \textbf{out}\ n.\mathit{done}[\textbf{out}\ b]]]] \mid - \ .$$

It remains to show that these contexts can indeed be used to observe ticks (respectively other actions for correspondingly defined contexts). This proof obligation has two sides, the first captured in Lemma 2. It stipulates that the tick-observing context put together with a tick-emitting system does observe the tick in that both reduce to a configuration containing the ambient *done* at top-level; the latter is used as marker to witness this successful interaction. The reverse property in Lemma 3 makes sure that, with the help of barbs, a context $\mathcal{C}_{n.\texttt{tick}}$ (or $\mathcal{C}_\alpha$ in the general cases) assures that the system under observation does make a $\texttt{tick}$-step (resp. α-step).

Note that all top level ambients of the system which is entered in the context will receive time via τ-actions. With this context, we can extend Lemma 4.8 of Merro and Zappa Nardelli [110] to virtually timed ambients as follows:

Lemma 2. *Let α be an observable, non-anonymous label and M a system. Then for all clocks* CLOCK *and processes P, if $M \xrightarrow{\alpha} M'$ then $\mathcal{C}_\alpha[M] \bullet (\text{CLOCK} \mid P) \Longrightarrow \simeq_s (M' \bullet (\text{CLOCK} \mid P)) \mid \mathit{done}[]$.*

Proof. We will only consider $\alpha = n.\texttt{tick}$. All other cases are similar to the ones for untimed mobile ambients. Let P be a process. We know that $M \xrightarrow{n.\texttt{tick}} M'$. Then $M' \equiv n[\texttt{tick} \mid Q] \mid M''$. Now,

$$\begin{aligned}
&\mathcal{C}_{n.\texttt{tick}}[M] \bullet (\text{CLOCK} \mid P)\\
\equiv&((\nu a, b)a[\textbf{in}\ n.\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\ \textbf{out}\ n.\mathit{done}[\textbf{out}\ b]])]] \mid M)\\
&\bullet (\text{CLOCK} \mid P)\\
\rightarrowtriangle&((\nu a, b)n[a[\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\ \textbf{out}\ n.\mathit{done}[\textbf{out}\ b]]]] \mid Q] \mid M'')\\
&\bullet (\text{CLOCK} \mid P)\\
\rightarrowtriangle&((\nu a, b)n[a[] \mid \texttt{tick}[b[\textbf{out}\ \texttt{tick}.\ \textbf{out}\ n.\mathit{done}[\textbf{out}\ b]]] \mid Q] \mid M'')\\
&\bullet (\text{CLOCK} \mid P)\\
\rightarrowtriangle&((\nu a, b)n[a[] \mid \texttt{tick} \mid Q] \mid b[\mathit{done}[\textbf{out}\ b]] \mid M'') \bullet (\text{CLOCK} \mid P)
\end{aligned}$$

$$\begin{aligned}&\rightarrowtriangle ((\nu a, b)n[a[] \mid \texttt{tick} \mid Q] \mid b[] \mid \mathit{done}[] \mid M'') \bullet (\textsc{Clock} \mid P)\\ &\equiv (M' \bullet (\textsc{Clock} \mid P)) \mid \mathit{done}[].\end{aligned}$$ □

To prove the correspondence between actions α and their contexts $\mathcal{C}_\alpha[-]$, we have to prove the converse of the above lemma as well. The proof of this result uses particular contexts $\texttt{spy}_\alpha\langle i, j, -\rangle$ as a technical tool to guarantee that the process P provided by the environment does not perform any action. Define the context $\texttt{spy}_{n.\texttt{tick}}\langle i, j, -\rangle := (i[] \mid -) \oplus (j[] \mid -)$, where $\oplus$ represents (an encoding of) the internal choice between the left and right process. We can now extend Lemma 4.12 of [110] as follows:

Lemma 3. *Let α be an observable, non-anonymous action and M a system. Let i and j be fresh names for M. For all processes P with $\{i, j\} \cap fn(P) = \emptyset$, if $C_\alpha[M] \bullet (\textsc{Clock} \mid \mathit{spy}_\alpha\langle i, j, P\rangle) \Longrightarrow \equiv N \mid \mathit{done}[]$ and $N \Downarrow_i$ and $N \Downarrow_j$, then there exists a system M' such that $M \xrightarrow{\alpha} M'$ and $M' \bullet (\textsc{Clock} \mid \mathit{spy}_\alpha\langle i, j, P\rangle) \simeq_s N$.*

Proof. We only show the case of $\alpha = n.\texttt{tick}$. All other cases are similar to the cases for mobile ambients.

$$\begin{aligned}&\mathcal{C}_\alpha[M] \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i, j, P\rangle)\\ &\equiv (\nu a, b)a[\textbf{in}\ n.\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\ \textbf{out}\ n.\mathit{done}[\textbf{out}\ b]]]] \mid\\ &\quad M \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i, j, P\rangle)\\ &\rightarrowtriangle (\nu a, b)n[a[] \mid \texttt{tick} \mid Q] \mid b[] \mid M''\\ &\quad \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i, j, P\rangle) \mid \mathit{done}[] \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i, j, P\rangle)\\ &\equiv M' \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i, j, P\rangle) \mid \mathit{done}[] \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i, j, P\rangle)\\ &\equiv N \mid \mathit{done}[]\ .\end{aligned}$$

We conclude that $M \xrightarrow{n.\texttt{tick}} M'$. Thus, $M' \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i, j, P\rangle) \simeq_s N$ holds. □

Theorem 2. *Weak timed bisimilarity and reduction barbed congruence over timed systems coincide.*

Proof. By Theorem 1, $\approx^t$ is preserved by contexts. The relation is additionally preserved under barbs and under reductions, two properties which carry over from (untimed) mobile ambients. With $\simeq_s$ defined as the *largest* congruence enjoying those preservation properties, immediately $\approx^t \subseteq \simeq_s$. For the reverse inclusion, we can now extend Theorem 4.14 in [110] by means of Lemmas 2 and 3 above. The cases for the anonymous actions are the same as for the mobile ambients. Hence, reduction barbed congruence and weak timed bisimilarity coincide, i.e. $M \approx^t N$ iff $M \simeq_s N$. □

6.5 Relaxation over Time

It is possible that two virtually timed systems behave the same for a certain type of requests but not for others, or that a system behaves equivalently to another except for needing some extra time slices per executed request. Example 5 shows that the

timed behavior of a system depends on the kind of requests that are made, the number of virtually timed ambients which are used as computing environments, their speeds and the way the requests are distributed to the computing environments.

6.5.1 Bounded Bisimulation

To compare systems only for certain types of requests and to consider if one system is faster or slower in executing these requests, we define a bisimulation which does not consider all processes P but only specific ones and also relaxes the time condition. First we introduce an asymmetric order relation, a *simulation* which allows to compare systems of different "speed", but otherwise equivalent behavior.

Definition 15 (Weak k-timed $\mathcal{P}$-simulation)**.** *Let $\mathcal{P}$ be a class of processes and $k \in \mathbb{N}$. A relation $\mathcal{R}$ over systems is a* weak k-timed $\mathcal{P}$-simulation *if $M \mathcal{R} N$ and $M \xrightarrow{\alpha} M'$ implies*

1. *If α is a non-anonymous label, then $N \xrightarrow{m.\mathtt{tick}^i} \xrightarrow{\hat{\alpha}} \xrightarrow{m.\mathtt{tick}^j} N'$ for some N' and where $i+j \leq k$, such that $M'\bullet(\text{CLOCK} \mid P) \;\; \mathcal{R} \;\; N'\bullet(\text{CLOCK} \mid P)$ (for all $P \in \mathcal{P}$ and all* CLOCK*).*
2. *For anonymous labels:*
   (a) *If $\alpha = *.\mathtt{enter}\_n$, then $N \mid n[\circ] \xrightarrow{m.\mathtt{tick}^j} N'$ for some N' and were $j \leq k$. such that $M'\bullet(\text{CLOCK} \mid P) \;\; \mathcal{R} \;\; N'\bullet(\text{CLOCK} \mid P)$, (for all $P \in \mathcal{P}$ and all* CLOCK*).*
   (b) *If $\alpha = *.\mathtt{exit}\_n$, then $n[\circ \mid N] \xrightarrow{m.\mathtt{tick}^j} N'$ for some N' and were $j \leq k$. such that $M'\bullet(\text{CLOCK} \mid P) \;\; \mathcal{R} \;\; N'\bullet(\text{CLOCK} \mid P)$, (for all $P \in \mathcal{P}$ and all* CLOCK*).*

Here $m.\mathtt{tick}^j$ denotes the j-fold execution of the $m.\mathtt{tick}$ action. A system M is *weakly k-timed $\mathcal{P}$-simulated* by N, written $M \preceq^t_{k,\mathcal{P}} N$, if $M \mathcal{R} N$ for some k-timed $\mathcal{P}$-simulation and some class of processes $\mathcal{P}$. Let k_{min} be the minimum k such that $M \preceq^t_{k,\mathcal{P}} N$ holds. This means that N needs at least k_{min} more tick steps than M to execute single requests of type $\mathcal{P}$. We will consider all further k to be the minimal k_{min} and call k the *slack* of the simulation.

Definition 16 (Weak k-timed $\mathcal{P}$-bisimulation)**.** *Let $\mathcal{P}$ be a class of processes and $k \in \mathbb{N}$. A symmetric weak k-timed $\mathcal{P}$-simulation $\mathcal{R}$ over systems is a* weak k-timed $\mathcal{P}$-bisimulation.

Systems M and N are *weakly k-timed $\mathcal{P}$-bisimilar*, written $M \approx^t_{k,\mathcal{P}} N$, if $M \mathcal{R} N$ for some k-timed $\mathcal{P}$-bisimulation $\mathcal{R}$ and some class of processes $\mathcal{P}$. This means that the systems need at least k additional tick steps to simulate each other's behavior. If $M \approx^t_{0,\mathcal{P}} N$ for a class of processes $\mathcal{P}$ then the systems are not distinguishable for these processes. Observe that if $M \approx^t_{0,\mathcal{P}} N$ for $\mathcal{P}$ the class of all processes then $M \approx^t N$. This follows from the definitions. We can make a similar observation about weak bisimulation without time.

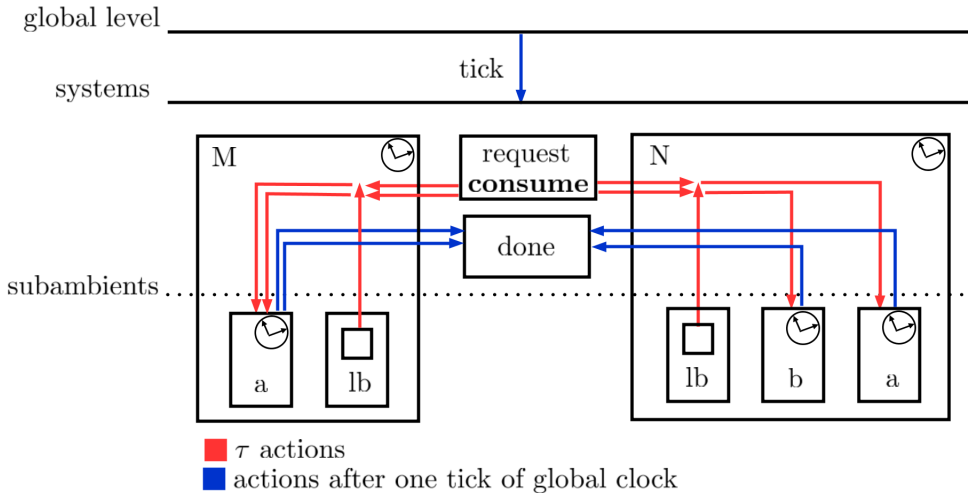

Figure 6.1: Indistinguishable behavior with round-robin load balancers and requests which need one computing resource to execute.

Lemma 4. *Let $\mathcal{P}$ be the class of all processes. Then $M \approx^t_{\infty,\mathcal{P}} N$ if and only if $M \approx N$.*

Proof. Assume $M \approx N$. This means, we consider a setting where we do not observe the ticking of the clocks but interpret all `tick`-actions as τ-actions, instead. Treating all `tick`-actions as τ-actions is equivalent to setting $k = \infty$ in the definition of k-timed $\mathcal{P}$-bisimilarity. Thus, $M \approx^t_{\infty,\mathcal{P}} N$ for $\mathcal{P}$ the class of all processes.

Assume now $M \approx^t_{\infty,\mathcal{P}} N$ for $\mathcal{P}$ the class of all processes. This is equivalent to treating `tick`-actions as unobservable τ-actions, thus $M \approx N$ in a setting where we do not observe the ticking of the clocks but interpret all `tick`-actions as τ-actions. □

We illustrate the concepts of weakly k-timed $\mathcal{P}$-bisimulation and slack in the following example.

Example 7 (Slack between two systems)**.** *Consider the two systems M and N from Example 5. System M has a clock with speed $(1, 1)$, a load balancer and the computing environment $a[\text{CLOCK}^{2,1} \mid \cdots]$, and further system N has a clock with speed $(2, 1)$, another load balancer and the computing environments $a[\text{CLOCK}^{1,1} \mid \cdots]$ and $b[\text{CLOCK}^{1,1} \mid \cdots]$. As M has only one virtually timed subambient, which is used as a computing environment, with a speed that equals the combined speeds in N, it holds that M can always simulate N; thus, $N \preceq^t_{0,\mathcal{P}} M$ for all $\mathcal{P}$. On the other hand the number of additional tick steps needed by N to simulate M depends on the load balancer in the system. If we assume that the load balancing is round-robin then the two systems are not distinguishable for processes which only need one computing resource to execute. An example can be seen in Figure 6.1. Thus, $M \approx^t_{0,\mathcal{P}} N$ for the class $\mathcal{P}$ of processes which need only one computing resource to execute. Since tasks which need more than one resource can not be served by N in one time slice it is easy to see that M will always be faster. This effect can be seen in Figure 6.2.*

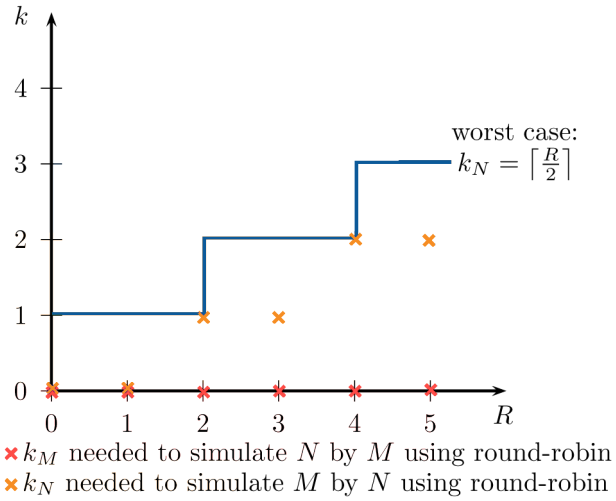

Figure 6.2: Worst case function for k_N as well as the actual k needed when using round-robin, where R is the maximal number of resources required by a single request.

6.5.2 Comparing Different Schedulers

In Example 7, round-robin is the best way to schedule the processes for distribution into the virtually timed subambients. But in general round-robin does not guarantee the fastest execution.

Definition 17. *We call the scheduling of processes* perfect *if the processes are always distributed in such a way that the execution takes the least possible amount of time.*

Note that building such a load balancer is a non-trivial optimisation problem. We use this definition to generalize the upper bound for the slack k_N in Example 7. Let $\mathcal{P}_\varnothing$ denote the class of all processes which do not require resources and therefore are not time consuming. If $M \approx^t_{0,\mathcal{P}_\varnothing} N$, then the systems do not have any internal processes which require a different amount of `tick` steps. Additionally we use the notion of a *statically deployed system*. In a statically deployed system, as for instance in Example 1, the time consuming virtually timed ambients do not move.

Definition 18. *A* statically deployed system *is a timed system where only virtually timed ambients with a speed of* $(0, 0)$ *are permitted to make use of the enter- and exit-capabilities.*

Lemma 5. *Let M and N be statically deployed systems. Assume $M \approx N$ and $M \approx^t_{0,\mathcal{P}_\varnothing} N$. Let (a_m, b_m) be the accumulated speed in a virtually timed ambient m, which is used as computing environment. Let $P \in \mathcal{P}$ be a parallel composition of independent requests $p_1, \ldots, p_n$, $n \in \mathbb{N}$, which require a number of computing resources r_{p_i}, $i \in \{1, \ldots n\}$, and result in observable changes to the system. Let $R = \max\{\sum_{i=1}^{n} r_{p_i} \mid P \in \mathcal{P}\}$ be the maximum of all required computing resources by any request. Then it holds that*

$M \preceq^t_{k,\mathcal{P}} N$ for some $\mathcal{P}$ and it holds that

$$k \leq \left\lceil \frac{R - \left\lceil \frac{R}{\sum_{m \in M} \frac{a_m}{b_m}} \right\rceil \cdot \min\{\frac{a_m}{b_m} \mid m \in N\}}{\min\{\frac{a_m}{b_m} \mid m \in N\}} \right\rceil .$$

Proof. As $M \approx N$, it holds that $M \preceq^t_{\infty,\mathcal{P}} N$ for all $\mathcal{P}$. As $M \approx^t_{0,\mathcal{P}_\varnothing} N$, there are no internal processes which influence the timing in M and N in different ways. As the scheduling functions of the systems are not known, the worst case scenario in terms of speed occurs if all requests end up in the slowest virtually timed subambient, which is used as a computing environment, in N, while all virtually timed ambients in M can be used perfectly. In this case the number of time steps needed by M to execute all requests is $\left\lceil \frac{R}{\sum_{m \in M} \frac{a_m}{b_m}} \right\rceil$. By the definition of k-timed $\mathcal{P}$-simulation, N can execute k time steps after a process $P \in \mathcal{P}$ has entered the system. To execute all requests in the slowest virtually timed ambient, it has to hold that $R \leq \left(\left\lceil \frac{R}{\sum_{m \in M} \frac{a_m}{b_m}} \right\rceil + k \right) \cdot \min\{\frac{a_m}{b_m} \mid m \in N\}$. Thus, the minimal k in the worst case scenario is

$$k = \left\lceil \frac{R - \left\lceil \frac{R}{\sum_{m \in M} \frac{a_m}{b_m}} \right\rceil \cdot \min\{\frac{a_m}{b_m} \mid m \in N\}}{\min\{\frac{a_m}{b_m} \mid m \in N\}} \right\rceil .$$

As k can be less for other scheduling functions, the inequation follows. □

Note that if M and N are not statically deployed, a similar estimation can be made by considering the minimum of all possible accumulated speeds during the computation. However, such an estimation is much less accurate.

In the following example, we consider a load balancing strategy which is different from round-robin.

Example 8 (System with elastic scaling)**.** *We modify Example 5 of systems with load balancers by considering an elastically scaling system O which is able to react to the size of the requests.*

$$\begin{aligned}
&\textit{system } O\textit{:}\ (\nu\ \textit{scale}, \textit{default}, a, b)\ \textit{ess}[\textsc{Clock}^{1,1} \mid \textit{scale} \mid \textit{default}]\\
&\quad \textit{request:}\ \textit{request}[P.\textit{done_signal} \mid \textbf{in}\ \textit{ess.size_signal}.\ \textbf{open}\ \textit{move}]\\
&\quad \textit{scale:}\ \textit{scale}[!(\textit{size_signal_a}.\ \textbf{open}\ \textit{lock}_a.a[\cdots] \mid\\
&\qquad\qquad \textit{size_signal_b}.\ \textbf{open}\ \textit{lock}_b.b[\cdots])\\
&\qquad\qquad !\textit{lock}_a[x[]] \mid !\textit{lock}_b[y[]] \mid\\
&\qquad\qquad !(\textbf{open}\ x.\textit{move}[\textbf{out}\ \textit{scale}.\ \textbf{in}\ \textit{request}.\ \textbf{in}\ a] \mid\\
&\qquad\qquad\quad \textbf{open}\ y.\textit{move}[\textbf{out}\ \textit{scale}.\ \textbf{in}\ \textit{request}.\ \textbf{in}\ b])]\\
&\textit{ambient default:}\ \textit{default}[!\ \textbf{open}\ a \mid !\ \textbf{open}\ b]\\
&\quad \textit{ambient a:}\ a[\textsc{Clock}^{1,1} \mid \textbf{out}\ \textit{scale}.\ \textbf{open}\ \textit{request.wait_ for _done}.\\
&\qquad\qquad \textbf{in}\ \textit{default} \mid \textit{done}[\textbf{out}\ \textit{default}.\ \textbf{out}\ \textit{ess}]]
\end{aligned}$$

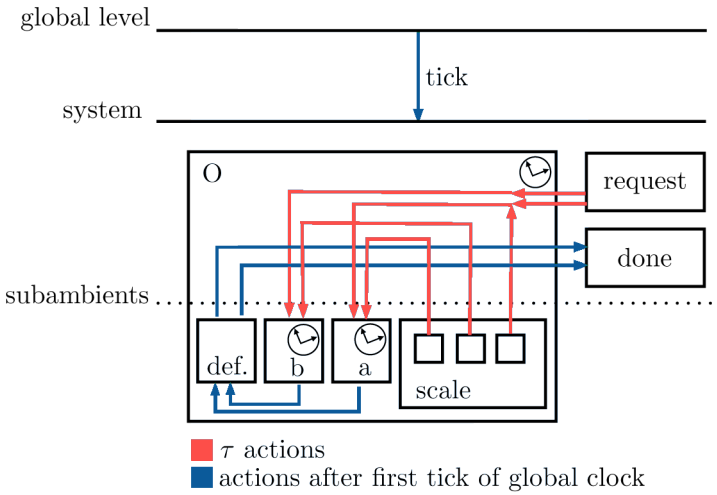

Figure 6.3: Example of a scaling system

ambient b: $b[\text{CLOCK}^{2,1}$ |**out** *scale*. **open** *request.wait_ for _done.*
in *default* | *done*[**out** *default*. **out** *ess*]]

In this example the ambient scale receives a signal from the request indicating its size. According to that scale releases ambients a or b with different speeds. The request is executed in this ambient and afterwards the ambient is deleted.

Figure 6.3 exemplifies the behavior of the system when given the requests

r_1*: request*[**c** . **c** *.done_signal* |
in *ess.size_signal_b*. **open** *move*] *and*
r_2*: request*[**c** *.done_signal* |**in** *ess.size_signal_a*. **open** *move*].

The given system shows the same behavior as the systems with load balancer in Example 5. At first glance the timed behavior seems similar to system M, but as this system can react to the size of the input while M uses a round-robin approach, it holds that $M \preceq^t_{0,\mathcal{P}} O$ *for all* $\mathcal{P}$*, while* $O \preceq^t_{k,\mathcal{P}} M$ *for some* $k \geq 0$.

If two systems release the same combined amount of resources in their virtually timed subambients for the same combined input value of time slices, we say they have the same *combined speed*. We can show that systems with the same combined speed are not distinguishable for small execution requests and perfect scheduling functions.

Lemma 6. *Assume* $M \approx N$ *where M and N do not have any internal processes which require a different amount of time steps. Assume the scheduling functions in M and N are perfect and M and N have the same combined speed in the virtually timed subambients which are used as computing environments. Let* $P \in \mathcal{P}$ *be a parallel composition of independent requests* $p_1, \ldots, p_n$*,* $n \in \mathbb{N}$*, which require a number of computing resources* r_{p_i}*,* $i \in \{1, \ldots n\}$*, and result in observable changes to the system. Then* $M \approx^t_{0,\mathcal{P}} N$ *if* $\mathcal{P}$ *is such that for all* $P \in \mathcal{P}$ *it holds that*

$$\max\{r_{p_i} \mid i \in \{1, \ldots n\}\} \leq 1 \ ,$$

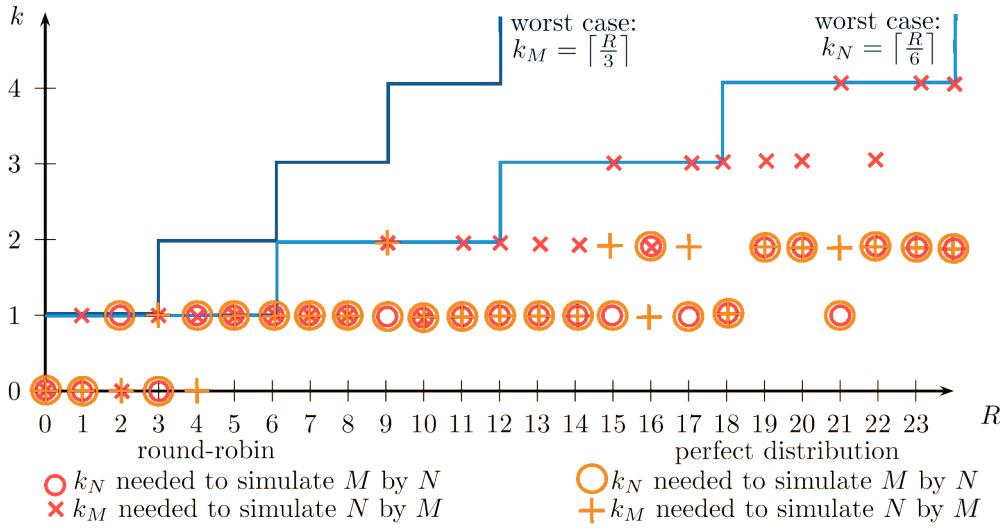

Figure 6.4: Worst case function and actual k needed by M and N in the case of round-robin and perfect scheduling, respectively.

i.e. the maximal requirement of resources in one request is less or equal one.

Proof. As the requests are at most of size one, are scheduled perfectly and both systems have the same combined speed, they will need the same number of time steps. □

In the following example we examine the worst case estimation of Lemma 5 with two different scheduling functions.

Example 9 (Worst case estimations)**.** *Consider two systems M and N, which are build similar to the system in Example 1. System M has a clock with speed $(2, 1)$, some load balancer and the ambients $a[\text{CLOCK}^{4,1} \mid \cdots]$ and $b[\text{CLOCK}^{2,1} \mid \cdots]$ for execution. System N has a clock with speed $(2, 1)$, some load balancer and the computing environments $a[\text{CLOCK}^{3,1} \mid \cdots]$ and $a[\text{CLOCK}^{3,1} \mid \cdots]$. As M and N have the same combined speed by Lemma 6, the systems are not distinguishable for perfect scheduling and requests of maximal size one. With round-robin scheduling , M needs $k = 1$ to simulate N for requests of size one. Figure 6.4 shows the worst case function as well as the actual slack k needed by the systems to simulate each other for different given scheduling functions and request sizes. The bigger the requests get, the worse the behavior of M with round-robin scheduling becomes. This is due to M containing the virtually timed subambient with the slowest local clock. Additionally we see that the slack k for perfect scheduling stays far behind the worst case estimation.*

6.6 Related Work

The extension of algebraic concurrency theories such as ACP, CCS and CSP to deal with time-dependent behavior can be done according to different design choices. Time can be absolute or relative. The time domain can be continuous, dense (so-called

real time), or discrete. Time can be semantically associated with regular actions, for instance via the use of timers, or it can be represented by special actions. Timed process algebras which originated from ACP and CSP can be found in, e.g., [14,15,122]. The model of virtually timed ambients developed in this paper builds on the mobile ambients. Therefore, we focus the discussion of related work on the π-calculus [146], which originated from CCS and which is closely related to the ambient calculus.

An early timed extension of CCS introduced a special action for time, without committing to a discrete or continuous time domain [118]. A related idling action σ is proposed in [77] such that processes in a standard process algebra would need exactly one time unit to process a σ, where time is discrete and processes synchronized via a global clock. A notion of local time for CCS is proposed in [147]; this notion resembles our model of local clocks, but was realized in terms of a timeout oriented model. A simulation-based faster-than preorder is introduced in [103]; this preorder is related to our notion of time relaxation but the faster-than preorder allows a process to delay by at most one time unit. In addition, the high-level idea in these works is very different: All these approaches focus on speed as the *duration* of processes, while in our approach with local clocks speed describes the *processing power* of a virtually timed ambient.

Timers have been studied for both the distributed π-calculus [20, 137] and for mobile ambients [8, 9, 53]. In this line of work, timers, which are introduced to express the possibility of a timeout, are controlled by a global clock. In membrane computing, the execution of each rule similarly takes exactly one time unit, as given by a global clock [131]. To overcome this restriction of membrane computing for modeling actual chemical reactions, each rule in timed P systems [48] has an associated integer representing the time needed to complete the execution of the rule. This resembles the timer approach on mobile ambients [8, 9, 53]. In contrast, the source clocks at the global system level in our work recursively control local clocks which define the execution power of the nested virtually timed ambients. Modeling timeouts is a straightforward extension of our work.

The enhancement of a process algebra with resources as primitives is studied in [100], where priorities are added to make processes sensitive to scheduling. A similar approach with an explicit scheduling concept is studied in [119]. In contrast, the only primitive in our approach is the ambient and the scheduling is fixed in the implementation of the resource distribution. Calculi, which are simpler to implement than the ambient calculus are studied in [23] and [52]. The Kell calculus described in [23] is based on the M-calculus and uses higher order communication. The δ-calculus in [52] uses synchronous movements in order to model distributed mobile real-time business applications. However, both calculi fail to preserve the simplicity of the ambient calculus. CPL [29] is a core language for defining cloud services and their deployment on cloud platforms which aims to enable statically safe service composition and custom implementations of cloud services. In contrast to our work on virtually timed ambients, time and performance are not been considered for CPL.

Cardelli and Gordon defined a labeled transition system for their mobile ambients [45], but no bisimulation. A bisimulation relation for a restricted version of mobile ambients, called mobile safe ambients, is defined in [109] and provides the basis for later

work. Barbed congruence for the same fragment of mobile ambients is defined in [157]. It is shown in [67] that name matching reduction barbed congruence and bisimulation coincide in the π-calculus. A bisimulation relation with contextual labels for the ambient calculus is defined in [120], but this approach is not suitable for providing a simple proof method. A labelled bisimulation for mobile ambients is defined by Merro and Nardelli [110], who prove that this bisimulation is equivalent to reduction barbed congruence and develop up-to-proof techniques. The weak timed bisimulation defined in this paper is a conservative extension of this approach.

6.7 Concluding Remarks

Virtualization opens for new and interesting foundational models of computation by explicitly emphasizing deployment and resource management. This paper introduces virtually timed ambients, a formal model of hierarchical locations of execution with explicit resource provisioning. 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. This way, the computing power of a virtually timed ambient depends on its location in the deployment hierarchy. To reason about timed behavior in this setting, we define weak timed bisimulation for virtually timed ambients as a conservative extension of bisimulation for mobile ambients, and show that the equivalence of bisimulation and reduction barbed congruence is preserved by this extension. We define timed relaxation as an "equivalent but slower" simulation relation, allowing speed deviation up to a bounded amount of time.

The calculus of virtually timed ambients opens for further, interesting research questions. One line of research is in statically controlling resource management, for example by means of behavioral types. Another line of research is in dynamically controlling resource management by means of resource awareness. This line of work is suggested by examples in this paper such as load balancers but could be enhanced by reflective resource capabilities allowing a process to influence its own deployment similar to virtualization APIs found in the context of cloud computing.

CHAPTER 10

An Analysis Framework for Virtualization

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.

10.1 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* [60, 83]. 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 [37]. *Nested virtualization* [73] is crucial to support cloud systems, as it enables virtual machines to migrate between different cloud providers [159]. It is also necessary to host virtual machines with operating systems which themselves support virtualization [19], such as Microsoft Windows 7 and Linux KVM.

Virtually timed ambients [89] is a calculus of explicit resource provisioning, based on the well-known calculus of mobile ambients [43]. 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 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 [90] 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 [90]. Maude was chosen as execution platform as it provides an intuitive way to model distributed systems at a high level of abstraction [125].

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 10.2. Section 10.3 describes a library of building blocks for cloud models in virtually timed ambients. Section 10.4 presents different examples of cloud architecture and analyses them using modal logic. We discuss related work and conclude in Sections 10.5 and 10.6.

10.2 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 [43] 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. 10.1.

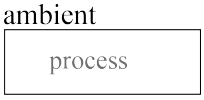

Figure 10.1: Graphical representation of a mobile ambient containing a process.

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. 10.2); 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 [43].

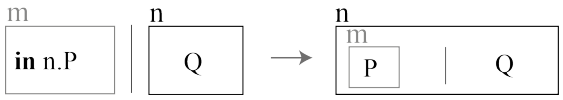

Figure 10.2: Graphical representation of the **in** n capability of mobile ambients.

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 [88, 89] 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. 10.3). 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.

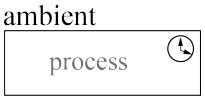

Figure 10.3: Graphical representation of a virtually timed ambient with a scheduler.

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.

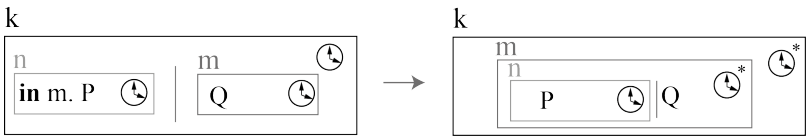

Figure 10.4: Graphical representation of the **in** m capability. The updated schedulers of the old and new parental ambient after the movement are marked with *.

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. 10.4). 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. 10.5).

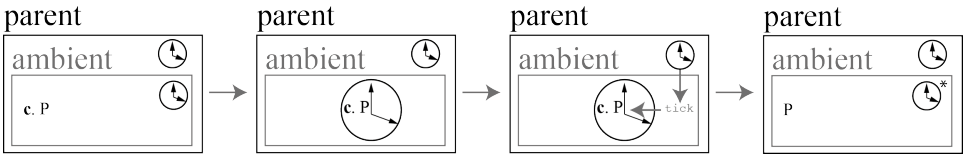

Figure 10.5: Graphical representation of the consume capability moving into the scheduler and consuming a resource after the ambient receives a `tick` from the parental ambient.

The calculus of virtually timed ambients has been implemented [90] 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 [89], 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.
```

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) .
```

[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`

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 [42] with notions based on metric temporal logic [96, 127, 128] to define a modal logic for virtually timed ambients [90]. 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.

10.3 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.

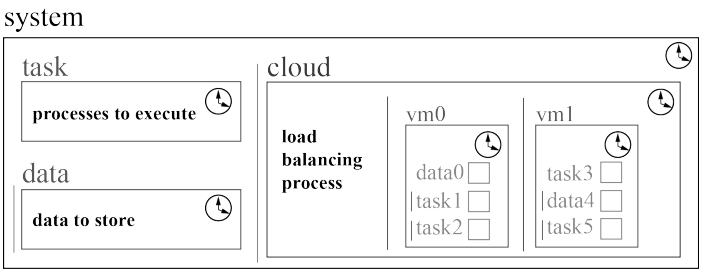

Figure 10.6: Example configuration of a cloud model.

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 [108]. 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] .
```

10.4 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 10.3, 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.

10.5 Related Work

Virtually timed ambients were first defined to study bisimulation for virtual resource management [88]. They are based on mobile ambients [43], which model location mobility for processes executing in distributed, hierarchical networks. Gordon proposed a simple formalism for virtualization loosely based on mobile ambients [74]. Virtually timed ambients [88,89] 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 [8] and in process calculi in general [14, 77, 118] 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 [42]. We combine this logic with ideas from metric temporal logic [96, 127, 128] to specify notions of time and resources [90].

The operational reduction rules for mobile ambients as well as a type system have been implemented in Maude before [142]. In contrast, our implementation focuses on capturing the timed reduction rules of virtually timed ambients as well as modal formulas to enable model checking.

10.6 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.

Bibliography

[1] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, 1995.

[2] Abdelzahir Abdelmaboud, Dayang N.A. Jawawi, Imran Ghani, Abubakar Elsafi, and Barbara Kitchenham. Quality of service approaches in cloud computing: A systematic mapping study. *Journal of Systems and Software*, 101:159 – 179, 2015.

[3] Erika Ábrahám, Immo Grabe, Andreas Grüner, and Martin Steffen. Behavioral interface description of an object-oriented language with futures and promises. *Journal of Logic and Algebraic Programming*, 78(7):491–518 (28 pages), 2009. Special issue with selected contributions of NWPT'07. The paper is a reworked version of an earlier UiO Technical Report TR-364, Oct. 2007.

[4] Erika Ábrahám, Andreas Grüner, and Martin Steffen. Dynamic heap-abstraction for open, object-oriented systems with thread classes (extended abstract). In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers: CiE 2006*, volume 3988 of *Lecture Notes in Computer Science*, pages 1–10 (10 pages). Springer, July 2006. A preliminary version has been included in the informal workshop proceedings of Cosmicah'05, as Queen Mary Technical Report RR-05-04, a longer version has been published as Technical Report 0601 of the Institute of Computer Science of the University Kiel, January 2006.

[5] Luca Aceto and Andrew D. Gordon. *Algebraic process calculi: The first twenty five years and beyond*. Citeseer, 2008.

[6] Ozan Akar. Model Checking Of Ambient Calculus Specifications Against Ambient Logic Formulas. Bachelor's thesis, Istanbul Technical University, 2009.

[7] Elvira Albert, Frank S. de Boer, Reiner Hähnle, Einar B. Johnsen, Rudolf Schlatte, S. Lizeth Tapia Tarifa, and Peter Y. H. Wong. Formal modeling and analysis of resource management for cloud architectures: An industrial case study using Real-Time ABS. *Journal of Service-Oriented Computing and Applications*, 8(4):323–339, 2014.

[8] Bogdan Aman and Gabriel Ciobanu. Mobile ambients with timers and types. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Proceedings 4th International Colloquium on Theoretical Aspects of Computing (ICTAC'07)*, volume 4711 of *Lecture Notes in Computer Science*, pages 50–63. Springer, 2007.

[9] Bogdan Aman and Gabriel Ciobanu. Timers and proximities for mobile ambients. In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *Proceedings 2nd International Symposium on Computer Science in Russia (CSR'07)*, volume 4649 of *Lecture Notes in Computer Science*, pages 33–43. Springer, 2007.

[10] Bogdan Aman and Gabriel Ciobanu. Timed mobile ambients for network protocols. In *International Conference on Formal Techniques for Networked and Distributed Systems*, pages 234–250. Springer, 2008.

[11] Amazon Web Services. Auto scaling user guide. `http://docs.aws.amazon.com/autoscaling/latest/userguide/as-dg.pdf`. Accessed: 2017-06-21.

[12] Torben Amtoft. Flow-sensitive type systems and the ambient calculus. *Higher-Order and Symbolic Computation*, 21(4):411–442, 2008.

[13] Jos C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005.

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

[15] Jos C. M. Baeten and Cornelis A. Middelburg. *Process Algebra with Timing*. Monographs in Computer Science. An EATSC series. Springer, 2002.

[16] Jos C. M. Baeten and W. Peter Weijland. Process algebra. *Cambridge Tracts in Theoretical Computer Science*, 18, 1990.

[17] W. F. Bauer. Computer design from the programmer's viewpoint. In *Papers and Discussions Presented at the December 3-5, 1958, Eastern Joint Computer Conference: Modern Computers: Objectives, Designs, Applications*, AIEE-ACM-IRE '58 (Eastern), pages 46–51, New York, NY, USA, 1958. ACM.

[18] Hans Bekić. Towards a mathematical theory of processes. In *Programming Languages and Their Definition*, pages 168–206. Springer, 1984.

[19] Muli Ben-Yehuda, Michael D. Day, Zvi Dubitzky, Michael Factor, Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman, and Ben-Ami 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.

[20] Martin Berger. *Towards Abstractions for Distributed Systems*. PhD thesis, University of London, Imperial College, 2004.

[21] Martin Berger and Nobuko Yoshida. Timed, distributed, probabilistic, typed processes. In *Asian Symposium on Programming Languages and Systems*, pages 158–174. Springer, 2007.

[22] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka. *Handbook of process algebra*. Elsevier, 2001.

[23] Philippe Bidinger and Jean-Bernard Stefani. The Kell calculus: Operational semantics and type system. In Elie Najm, Uwe Nestmann, and Perdita Stevens, editors, *Proceedings 6th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2003)*, volume 2884 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 2003.

[24] Patrick Blackburn, Johan van Benthem, and Frank Wolter. *Handbook of modal logic*, volume 3. Elsevier, 2006.

[25] Iovka Boneva and Jean-Marc Talbot. When ambients cannot be opened. In *International Conference on Foundations of Software Science and Computation Structures*, pages 169–184. Springer, 2003.

[26] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Quantitative analysis of real-time systems using priced timed automata. *Communications of the ACM*, 54(9):78–87, 2011.

[27] Chiara Braghin, Agostino Cortesi, Stefano Filippone, Riccardo Focardi, Flaminia L. Luccio, and Carla Piazza. Banana-a tool for boundary ambients nesting analysis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 437–441. Springer, 2003.

[28] Chiara Braghin, Agostino Cortesi, Riccardo Focardi, Flaminia L. Luccio, and Carla Piazza. Complexity of nesting analysis in mobile ambients. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 86–101. Springer, 2003.

[29] Oliver Bračevac, Sebastian Erdweg, Guido Salvaneschi, and Mira Mezini. CPL: A core language for cloud computing. In *Proceedings 15th International Conference on Modularity (MODULARITY 2016)*, pages 94–105. ACM, 2016.

[30] Patrice Brémond-Grégoire and Insup Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189(1-2):179–219, 1997.

[31] Linda Brodo. On the expressiveness of π-calculus for encoding mobile ambients. *Mathematical Structures in Computer Science*, pages 1–39, 2016.

[32] Linda Brodo, Pierpaolo Degano, and Corrado Priami. Reflecting mobile ambients into the π-calculus. In *International Workshop on Global Computing*, pages 25–56. Springer, 2003.

[33] Michele Bugliesi and Giuseppe Castagna. Secure safe ambients. In *ACM SIGPLAN Notices*, volume 36, pages 222–235. ACM, 2001.

[34] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Boxed ambients. In *International Symposium on Theoretical Aspects of Computer Software*, pages 38–63. Springer, 2001.

[35] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Access control for mobile agents: The calculus of boxed ambients. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(1):57–124, 2004.

[36] Nadia Busi and Gianluigi Zavattaro. On the expressiveness of movement in pure mobile ambients. *Electronic Notes in Theoretical Computer Science*, 66(3):22–36, 2002.

[37] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona 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.

[38] Jeffrey P. Buzen and U. O. Gagliardi. The evolution of virtual machine architecture. In *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, AFIPS '73, pages 291–299, New York, NY, USA, 1973. ACM.

[39] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility types for mobile ambients. In *Lecture Notes in Computer Science*, volume 1644, page 230239. Springer, July 1999.

[40] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient groups and mobility types. In *IFIP International Conference on Theoretical Computer Science*, pages 333–347. Springer, 2000.

[41] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Types for the ambient calculus. *Information and Computation*, 177(2):160–194, 2002.

[42] Luca Cardelli and Andrew 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.

[43] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[44] Luca Cardelli and Andrew D. Gordon. Logical properties of name restriction. In *Proceedings 5th International Conference on Typed Lambda Calculi and Applications (TLCA 2001)*, volume 2044 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2001.

[45] Luca Cardelli and Andrew D. Gordon. Equational properties of mobile ambients. *Mathematical Structures in Computer Science*, 13(3):371–408, 2003.

[46] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Comput. Surv.*, 17(4):471–523, December 1985.

[47] Giuseppe Castagna, Jan Vitek, and Francesco Zappa Nardelli. The seal calculus. *Information and Computation*, 201(1):1–54, 2005.

[48] Matteo Cavaliere and Dragos Sburlan. Time-independent P systems. In Giancarlo Mauri, Gheorghe Paun, Mario J. Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Proceedings 5th International Workshop on Membrane Computing (WMC'04)*, volume 3365 of *Lecture Notes in Computer Science*, pages 239–258. Springer, 2004.

[49] Witold Charatonik, Silvano Dal-Zilio, Andrew D. Gordon, Supratik Mukhopadhyay, and Jean-Marc Talbot. The complexity of model checking mobile ambients. In Furio Honsell and Marino Miculan, editors, *Proceedings of Foundations of Software Science and Computation Structures, 4th International Conference (FOSSACS 2001)*, volume 2030 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 2001.

[50] Witold Charatonik, Andrew D. Gordon, and Jean-Marc Talbot. Finite-control mobile ambients. In *European Symposium on Programming*, pages 295–313. Springer, 2002.

[51] Witold Charatonik and Jean-Marc Talbot. The decidability of model checking mobile ambients. In Laurent Fribourg, editor, *Proceedings of 15th International Workshop on Computer Science Logic (CSL 2001)*, volume 2142 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2001.

[52] Yeongbok Choe and Moonkun Lee. δ-calculus: Process algebra to model secure movements of distributed mobile processes in real-time business applications. In *23rd European Conference on Information Systems, ECIS 2015, Münster, Germany, May 26-29, 2015*, 2015.

[53] Gabriel Ciobanu. Interaction in time and space. *Electronic Notes in Theoretical Computer Science*, 203(3):5–18, 2008.

[54] Gabriel Ciobanu and Vladimir A Zakharov. Encoding mobile ambients into the π-calculus. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 148–165. Springer, 2006.

[55] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT press, 1999.

[56] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude — A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[57] European Commission. European cloud computing strategy. `https://ec.europa.eu/digital-agenda/en/european-cloud-initiative`, 2015. Accessed: 2018-05-21.

[58] 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 International Conference on Cluster Computing*, pages 378–385, Sept 2011.

[59] Maxwell John Cresswell and George Edward Hughes. *A new introduction to modal logic*. Routledge, 2012.

[60] Tharam S. Dillon, Chen Wu, and Elizabeth 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.

[61] Jin Song Dong, Jing Sun, Jun Sun, Kenji Taguchi, and Xian Zhang. Specifying and verifying sensor networks: An experiment of formal methods. In *International Conference on Formal Engineering Methods*, pages 318–337. Springer, 2008.

[62] Jérôme Feret. Abstract interpretation-based static analysis of mobile ambients. In *International Static Analysis Symposium*, pages 412–430. Springer, 2001.

[63] Gianluigi Ferrari, Eugenio Moggi, and Rosario Pugliese. Guardians for ambient-based monitoring. *Electronic Notes in Theoretical Computer Science*, 66(3):52–75, 2002.

[64] Fibonacci. Greedy algorithm for Egyptian fractions. Available at `https://en.wikipedia.org/wiki/Greedy_algorithm_for_Egyptian_fractions`, Accessed: 2017-12-08.

[65] Howard Foster, Wolfgang Emmerich, Jeff Kramer, Jeff Magee, David Rosenblum, and Sebastian Uchitel. Model checking service compositions under resource constraints. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 225–234. ACM, 2007.

[66] Cédric Fournet and Georges Gonthier. *The Join Calculus: A Language for Distributed Mobile Programming*. Springer, 2002.

[67] Cédric Fournet and Georges Gonthier. A hierarchy of equivalences for asynchronous calculi. *Journal of Logic and Algebraic Programming*, 63(1):131 – 173, 2005.

[68] Yuxi Fu. Fair ambients. *Acta Informatica*, 43(8):535–594, 2007.

[69] Pablo Garralda, Adriana Compagnoni, and Mariangiola Dezani-Ciancaglini. BASS: Boxed Ambients with Safe Sessions. In Michael Maher, editor, *PPDP'06*, pages 61–72. ACM Press, 2006.

[70] Elio Giovannetti. Ambient calculi with types: a tutorial. In *Global Computing - Programming Environments, Languages, Security and Analysis of Systems*, volume 2874 of *Lecture Notes in Computer Science*, pages 151–191. Springer, 2003.

[71] Jens C. Godskesen, Thomas Hildebrandt, and Vladimiro Sassone. A calculus of mobile resources. In Luboš Brim, Mojmír Křetínský, Antonín Kučera, and Petr Jančar, editors, *Proceedings 13th International Conference on Concurrency Theory (CONCUR 2002)*, volume 2421 of *Lecture Notes in Computer Science*, pages 272–287. Springer, 2002.

[72] Robert P. Goldberg. Architecture of virtual machines. In *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, AFIPS '73, pages 309–318, New York, NY, USA, 1973. ACM.

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

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

[75] René R. Hansen, Jacob G. Jensen, Flemming Nielson, and Hanne R. Nielson. Abstract interpretation of mobile ambients. In *International Static Analysis Symposium*, pages 134–148. Springer, 1999.

[76] Daniel Hausmann, Till Mossakowski, and Lutz Schröder. A coalgebraic approach to the semantics of the ambient calculus. *Theoretical computer science*, 366(1-2):121–143, 2006.

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

[78] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82 – 120, 2002.

[79] Charles A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[80] Kohei Honda and Nobuko Yoshida. Replication in concurrent combinators. In Masami Hagiya and John C. Mitchell, editors, *TACS*, volume 789 of *Lecture Notes in Computer Science*, pages 786–805. Springer, 1994.

[81] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.

[82] Atsushi Igarashi and Naoki Kobayashi. Resource usage analysis. *ACM Trans. Program. Lang. Syst.*, 27(2):264–313, 2005.

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

[84] Yosr Jarraya, Arash Eghtesadi, Mourad Debbabi, Ying Zhang, and Makan Pourzandi. Cloud calculus: Security verification in elastic cloud computing platform. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 447–454. IEEE, 2012.

[85] Zhang Jing, Zhang Li-Cui, and Guo De-Gui. Transformation from seal calculus to mobile ambient calculus. *International Journal of Computer Science and Network Security*, 6(5A):179, 2006.

[86] Einar B. Johnsen, Reiner Hähnle, Jan Schäfer, Rudolf Schlatte, and Martin Steffen. ABS: A core language for abstract behavioral specification. In Bernhard Aichernig, Frank S. de Boer, and Marcello M. Bonsangue, editors, *Proc. 9th International Symposium on Formal Methods for Components and Objects (FMCO 2010)*, volume 6957 of *Lecture Notes in Computer Science*, pages 142–164. Springer, 2011.

[87] Einar B. Johnsen, Rudolf Schlatte, and S. Lizeth 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.

[88] Einar B. Johnsen, Martin Steffen, and Johanna B. Stumpf. A calculus of virtually timed ambients. In Phillip James and Markus Roggenbach, editors, *Postproceedings of the 23rd International Workshop on Algebraic Development Techniques (WADT 2016)*, volume 10644 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2017.

[89] Einar B. Johnsen, Martin Steffen, and Johanna B. Stumpf. Virtually timed ambients: A calculus of nested virtualization. *Journal of Logical and Algebraic Methods in Programming*, 94:109 – 127, 2018.

[90] Einar B. Johnsen, Martin Steffen, Johanna B. Stumpf, and Lars Tveito. Checking modal contracts for virtually timed ambients, 2018. Submitted for publication. Available at `www.ifi.uio.no/~johanbst/MLCheckingVTAS.pdf`.

[91] Cliff B. Jones. Tentative steps towards a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, 1983.

[92] Toru Kato, Atom Miyai, and Masahiro Higuchi. Experiment of a freight management system with the multiple ambient calculus. In *Mathematics and Computers in Sciences and in Industry (MCSI), 2015 Second International Conference on*, pages 191–198. IEEE, 2015.

[93] Shin-ya Katsumata. Parametric effect monads and semantics of effect systems. In *Proceedings of POPL '14*, pages 633–645. ACM, 2014.

[94] Naoki Kobayashi and Davide Sangiorgi. A hybrid type system for lock-freedom of mobile processes. *ACM Trans. Program. Lang. Syst.*, 32(5):16:1–16:49, 2010.

[95] Naoki Kobayashi, Kohei Suenaga, and Lucian Wischik. Resource usage analysis for the π-calculus. *Logical Methods in Computer Science*, 2(3), 2006.

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

[97] Jeff Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, 2007.

[98] Leslie Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, 1983.

[99] Edward A. Lee. Computing needs time. *Communications of the ACM*, 52(5):70–79, 2009.

[100] Insup Lee, Anna Philippou, and Oleg Sokolsky. Resources in process algebra. *Journal of Logic and Algebraic Programming*, 72(1):98 –122, 2007.

[101] Jeremy Y. Lee and John Zic. On modeling real-time mobile processes. In *Australian Computer Science Communications*, volume 24, pages 139–147. Australian Computer Society, Inc., 2002.

[102] Francesca Levi and Davide Sangiorgi. Mobile safe ambients. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 25(1):1–69, 2003.

[103] Gerald Lüttgen and Walter Vogler. Bisimulation on speed: Worst-case efficiency. *Information and Computation*, 191(2):105–144, 2004.

[104] Sergio Maffeis and Iain Phillips. On the computational strength of pure ambient calculi. *Electronic Notes in Theoretical Computer Science*, 96:29–49, 2004.

[105] Dan C. Marinescu. *Cloud Computing*. Morgan Kaufmann, Boston, 2013.

[106] Kenneth L. McMillan. *Symbolic Model Checking*. Springer, 1993.

[107] Peter Mell, Tim Grance, et al. The NIST definition of cloud computing. *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg*, 2011. Available at `https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf`.

[108] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.

[109] Massimo Merro and Matthew Hennessy. A bisimulation-based semantic theory of safe ambients. *ACM Transactions on Programming Languages and Systems*, 28(2):290–330, 2006.

[110] Massimo Merro and Francesco Zappa Nardelli. Behavioral theory for mobile ambients. *Journal of the ACM*, 52(6):961–1023, 2005.

[111] José Meseguer. Twenty years of rewriting logic. *Journal of Logic and Algebraic Programming*, 81(7-8):721–781, 2012.

[112] José Meseguer and Grigore Rosu. The rewriting logic semantics project. *Theoretical Computer Science*, 373(3):213–237, 2007.

[113] Robin Milner. *A Calculus of Communicating Systems*. Springer, Secaucus, NJ, USA, 1982.

[114] Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[115] Robin Milner. The polyadic pi-calculus: a tutorial. Technical report, Logic and Algebra of Specification, 1991.

[116] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.

[117] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7:417–426, 1981.

[118] Faron Moller and Chris 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.

[119] Mohammad Reza Mousavi, Michel A. Reniers, Twan Basten, and Michel R. V. Chaudron. PARS: A process algebraic approach to resources and schedulers. In M. Alexander and W. Gardner, editors, *Process Algebra for Parallel and Distributed Processing*. Chapman and Hall/CRC, 2008.

[120] Masaki Murakami. Congruent bisimulation equivalence of ambient calculus based on contextual transition system. In *Proceedings 7th International Symposium on Theoretical Aspects of Software Engineering (TASE 2013)*, pages 149–152. IEEE, 2013.

[121] Uwe Nestmann. Welcome to the jungle: A subjective guide to mobile process calculi. In *CONCUR 2006–Concurrency Theory*, pages 52–63. Springer, 2006.

[122] Xaviet Nicollin and Joseph Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.

[123] Hanne Riis Nielson, Flemming Nielson, and Mikael Buchholtz. Security for mobility. In *International School on Foundations of Security Analysis and Design*, pages 207–265. Springer, 2001.

[124] Massachusetts Institute of Technology and P. A. Crisman. *The compatible time-sharing system : a programmer's guide*. M. I. T. Press Cambridge, 2. edition, 1965.

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

[126] Dominic Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. In *POPL 2016*. ACM Press, 2016.

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

[128] Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In Franck Cassez and Claude 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.

[129] David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science*, pages 167–183, Berlin, Heidelberg, 1981. Springer.

[130] Douglas F. Parkhill. *The Challenge of the Computer Utility*. The Challenge of the Computer Utility. Addison-Wesley Publishing Company, 1966.

[131] Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[132] Tomas Petricek, Dominic Orchard, and Alan Mycroft. Coeffects: unified static analysis of context-dependence. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP'13)*, volume 7966 of *Lecture Notes in Computer Science*, pages 385–397. Springer, 2013.

[133] Tomas Petricek, Dominic Orchard, and Alan Mycroft. Coeffects: A calculus of context-dependent computation. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the International Conference on Functional Programming (ICFP'14)*. ACM, 2014.

[134] Iain Phillips and Maria Grazia Vigliotti. On reduction semantics for the push and pull ambient calculus. In *Foundations of Information Technology in the Era of Network and Mobile Computing*, pages 550–562. Springer, 2002.

[135] Iain Phillips and Maria Grazia Vigliotti. Electoral systems in ambient calculi. In *International Conference on Foundations of Software Science and Computation Structures*, pages 408–422. Springer, 2004.

[136] Benjamin C. Pierce. *Types and programming languages*. MIT press, 1st edition, 2002.

[137] Cristian Prisacariu. Timed distributed pi-calculus. In *Modelling and Verifying of Parallel Processes (MOVEP06)*, pages 348–354, 2006.

[138] John Patrick Pullen. Where did cloud computing come from, anyway? `http://time.com/collection-post/3750915/cloud-computing-origin-story/`. Accessed: 2018-06-04.

[139] David Pym and Chris Tofts. A calculus and logic of resources and processes. *Formal Aspects of Computing*, 18(4):495–517, 2006.

[140] George M. Reed and A. William Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58(1-3):249–261, 1988.

[141] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. Bioambients: an abstraction for biological compartments. *Theoretical Computer Science*, 325(1):141–167, 2004.

[142] Fernando Rosa-Velardo, Clara Segura, and Alberto 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.

[143] Davide Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, 1996.

[144] Davide Sangiorgi. Extensionality and intensionality of the ambient logics. *SIGPLAN Not.*, 36(3):4–13, 2001.

[145] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

[146] Davide Sangiorgi and David Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[147] Ichiro Satoh and Mario Tokoro. A timed calculus for distributed objects with clocks. In Oscar M. Nierstrasz, editor, *Proceedings 7th European Conference on Object-Oriented Programming (ECOOP'93)*, pages 326–345. Springer, 1993.

[148] Ichiro Satoh and Mario Tokoro. Time and asynchrony in interactions among distributed real-time objects. In *European Conference on Object-Oriented Programming*, pages 331–350. Springer, 1995.

[149] Alan Schmitt and Jean-Bernard Stefani. The m-calculus: A higher-order distributed process calculus. In *ACM SIGPLAN Notices*, volume 38, pages 50–61. ACM, 2003.

[150] François Siewe, Hussein Zedan, and Antonio Cau. The calculus of context-aware ambients. *Journal of Computer and System Sciences*, 77(4):597 – 620, 2011.

[151] Eugene W. Stark. A proof technique for rely/guarantee properties. In S. N. Maheshwari, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 206 of *Lecture Notes in Computer Science*, pages 369–391. Springer, 1985.

[152] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: Towards flexible verification under fairness. In *Proc. 21th International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.

[153] Yujie Sun. Toward a Model Checker for Ambient Logic Using the Process Analysis Toolkit. Master's thesis, Bishop's University, Sherbrooke, Quebec, Canada, 2015.

[154] David Teller, Pascal Zimmer, and Daniel Hirschkoff. Using ambients to control resources. In *Proceedings of the 13th International Conference on Concurrency Theory*, CONCUR '02, pages 288–303, London, UK, 2002. Springer.

[155] Devrim Unal, Ozan Akar, and M. Ufuk Caglayan. Model checking of location and mobility related security policy specifications in ambient calculus. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 155–168. Springer, 2010.

[156] Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. *Electronic Notes in Theoretical Computer Science*, 203:263–284, 2008. Proceedings 9th Intl. Workshop on Coalgebraic Methods in Computer Science (CMCS 2008).

[157] Maria Grazia Vigliotti and Iain C.C. Phillips. Barbs and congruences for safe mobile ambients. *Electronic Notes in Theoretical Computer Science*, 66(3):37 – 51, 2007.

[158] Philipp Wieder, Joe M. Butler, Wolfgang Theilmann, and Ramin Yahyapour. *Service level agreements for cloud computing*. Springer Science & Business Media, 2011.

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

[160] Franco Zambonelli, Giacomo Cabri, and Letizia Leonardi. Developing mobile agent organizations: A case study in digital tourism. In *Distributed Objects and Applications, 2001. DOA'01. Proceedings. 3rd International Symposium on*, pages 270–279. IEEE, 2001.

List of Figures

List of Tables