Characterization and Analysis of Flexible Energy Prosumers using Resource Timed Automata

Jonas Hansen
Computer Science, 2020-2022

4+4 PHD Part A, Master's Project

The report is written in LaTeX. Figures generated by **Tikz** and **UPPAAL**. Bibliography generated by **Bibtex**.

Department of Computer Science
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Characterization and Analysis of Flexible Energy Prosumers using Resource Timed Automata

Theme:
Modeling and Verification

Project Period:
August 2020 - August 2022

Participant(s):
Jonas Hansen

Supervisor(s):
Kim Guldstrand Larsen

Copies: 1

Page Numbers: 22

Date of Completion:
August 10, 2022

Abstract:

We address the problem of modeling and analyzing the flexible behavior of Prosumer systems for the purpose of balancing production and consumption of energy in, for example, an energy grid. We introduce the notion of Resource Timed Automata (RTA) and show how prosumers can be modeled using these automata. Furthermore, we show that aggeregation of prosumers in a smart electricity grid can be modeled as a special kind of parallel composition of such automata, and the balancing problem can be formalized as reachability problems on an observer automaton. Finally, we illustrate how these reachability problems can be solved using established methods from model checking, and discuss why we suspect these methods will scale well in practice for this type of problem.

Contents

Preface

This report serves as the primary artifact for which the author will defend his Master's degree as part of the 4+4 PHD programme at the Department of Computer Science, Aalborg University, DEIS, under the supervision of Professor Kim Guldstrand Larsen. It details one of the projects worked on during Part A of the programme.

The main body of the report details and extends the work **Balancing Flexible Production and Consumption of Energy using Resource Timed Automata**[6] published in the proceedings of the *11th Mediterranean Conference on Embedded Computing Resources* (MECO22), 7-10 June, Budva, Montenegro. Originally [6] is written by *Jonas Hansen*, *Kim Guldstrand Larsen* and *Pieter J.L Cuijpers*.

Aalborg University, August 10, 2022

Jonas Hansen
<jonash@cs.aau.dk>

Chapter 1

Introduction

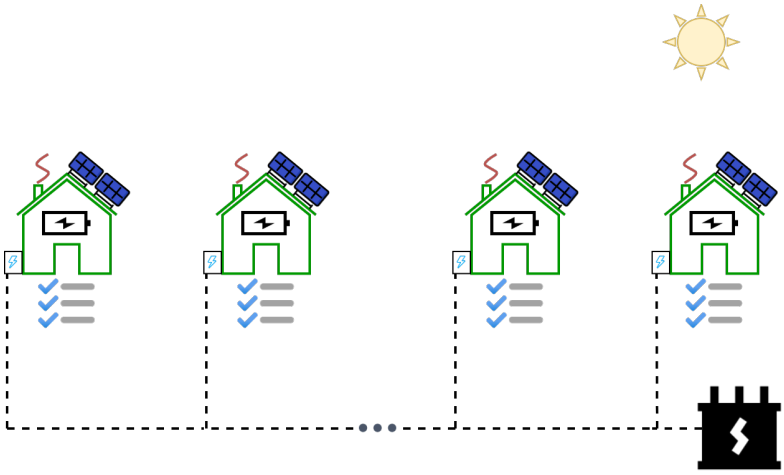


Figure 1.1: Example of a prosumer system. A number of households each associating a battery, photovoltaic solar panels and a list of tasks to complete. Households can choose to draw power from the grid and/or from its solar panels which in itself is dependent on the weather and the time of day.

In recent years we have seen an increase in decentralized renewable energy installations, such as rooftop solar panels, private windmills, heatpumps and onsite energy storage devices. Households, factories, office buildings etc. utilizing these installations are collectively referred to as prosumers, i.e. flexible electrical components (composites of components) both capable of producing and consuming power depending on time and/or non-deterministic behavior (see figure 1.1). Prosumers belong to the more general category of *Resource Systems* in which timed behavior alters some kind of observable hybrid resources, referred to as *resource behavior*. In general, such systems are dependent on their resource behavior, i.e. they are capable of testing their resources in order to alter their timed behavior. In this project we limit ourselves to resource observers in which no testing is possible during execution. We call this subset of resource systems under said constraints *Prosumers* or *Networks of Prosumers*. In addition to resource constraints, prosumers are also subject to task dependencies, meaning they must complete certain tasks (which might be

dependent on each other) in a finite amount of time.

An important challenge for a prosumer network is to balance the production and consumption of energy both locally and globally. This is called load management. Poor load management increases the strain on the power transition infrastructure, which in turn increases the risk of damage to that infrastructure as well as rolling blackouts resulting from voltage fluctuations. Good load management means to ensure that all power produced in a network is either immediately consumed at a nearby location or is stored locally as energy for future use [14].

Two complementary approaches for load management are energy storage in case of overproduction, and flexible scheduling of consumption and (sometimes) production. In traditional power grids, designed to enable consumers to draw power from producers in a one directional transfer, storage is usually achieved by storing excess power in the grid itself or by passive participation of customers in the form of "dumb" storage, e.g. in batteries of electrical vehicles. Furthermore, flexible scheduling is traditionally achieved through pricing strategies, that offer lower prices to customers who adapt their consumption to match optimal production efficiency, e.g. day/night tariffs and special contracts with industries [4]. Modern power grids, also known as smart grids, are designed for bi-directional transfer of power because of the presence of prosumers next to the traditional producers and consumers. This facilitates intelligent integration of collective behavior in terms of consumption/production. Varying sizes, output and capacities of energy producing components are more easily facilitated as well. Of particular note, smart grids allow for active participation of prosumers in order to optimize grid operations, but this requires negotiations between prosumers and the grid aggregators that are responsible for (parts of) the network [8].

In order to facilitate the scheduling of consumption and production by prosumers, the MIRABEL Project [11] introduced the notion of a *flex-offer*, in which a prosumer specifies upper and lower bounds to the energy profile requested for the duration of a given task, and offers a flexible start-time for that task to be scheduled. The MIRABEL Project also discussed how to use flex-offers as a basis for setting up contracts for the consumption and production of energy among prosumers and grid aggregators, but the details of this are outside the scope of this project. The original definition of flex-offer is quite restricted in what a prosumer can express as an energy profile, in later work the flexibility models have been generalized to include, in particular, state-based dependencies and linear time-invariant upper and lower bounds. After extending the notion of flex-offer, the next step is usually to study the problem of parallel composition through aggregation. By aggregating a number of flex-offers and determining the worst- and best-case energy profile for the aggregation, one can obtain insight in the baseline energy imbalance for a given scheduling of prosumers. However, this does not yet solve the scheduling problem itself [3, 10, 12, 13, 15, 16] and neither does it answer whether a proposed flexoffer is satisfied by a given prosumer.

In this project, we propose to model prosumer behavior using a variant of Hybrid Automata [7], named Resource Timed Automata (RTA). The novelty of this type of modeling, is that it allows a grid aggregator to consider prosumers in isolation, but also treat them as smaller or larger groups, because aggregation (i.e. parallel composition) of RTA's always results in another RTA. It is our expectation that abstraction techniques on RTA's can be developed in a similar fashion as for other types of automata, so that analysis methods for RTA's become scaleable. As an immediate contribution, we show in this project how to

answer the question whether an RTA satisfies a given flex-offer as a reachability problem, using standard methods from model-checking such as the UPPAAL toolset[5], and note that the witnessing trace produced by these model-checkers is in fact such a schedule. Before we dive into the proposed method and formalism we spent some time on formally defining so called Resource Systems as a generalization for the particular hybrid behavior capturing the semantics of consumers, producers and prosumers.

The remainder of this report is structured as follows. In Chapter 2 we formally define Resource Systems through its semantics and derive the relevant flexibility properties of interest. Chapter 3 formally defines Resource Timed Automata (RTA) and the parallel composition of such. Additionally we show how the semantics of such automata indeed captures the flexibility properties defined previously. Chapter 4 Defines the analysis method and discusses its merits. Lastly, Chapter 5 concludes upon the main body of the report and discusses ongoing/future projects.

Chapter 2

Resource Systems and Behavior

We now define prosumers and flexible energy behavior. We start by describing a general notion of resource systems and reason how prosumers are a subset of this overall intuition. We then proceed to define desirable flexibility properties and show how concepts of previous literature are defined in this setting. Additionally, we introduce the properties on which the analysis method is based on.

Intuitively, *Resource systems* are simply time dependent cyber-physical systems that observes or alters some resource such as water, electricity, pressure etc. We refer to this behavior as *resource behavior*. This intuition is based on an extension to the algebraic framework developed in [1] for timed regular languages. Flexibility analysis is the process of verifying whether a given resource behavior satisfies some property. In general, such systems can test their observations and act accordingly when running, however we limit ourselves to strictly observational behavior and also to time bound properties for our analysis. These restrictions are enforced because otherwise we lose a great deal of general decidability which is outside the scope of this particular project.

The system types; Consumer, Producer and Prosumer are simply systems that semantically adhere to some resource behavior resulting in negative, positive or mixed flow, respectively. As such, system types are only important when analysis is ongoing an essentially only applicable to instances of system behavior, not a strict label. Because of this, we refer to all systems as prosumers and moving forward we are only concerned with their resource flow behavior.

Moving forward we focus on electrical component defined over the resource semi-ring $(\mathbb{R}, +, \cdot, 0, 1)$ (where $+$ and $\cdot$ defines the arithmetic operations; addition and multiplication) and refer to instances[1] of the flow induced by electrical components as *power* and characterize it using functions of power over time $P(t) : \mathbb{R}_{\geq 0} \to \mathbb{R}$. In the remainder of this chapter we define flexibility properties in relation to the resource semi-ring. Note that we do not in fact require the resource domain to capture a semi-ring, however in chapter 3 we implicitly define our abstraction over this semi-ring. For convenience we therefore introduce it now.

Definition 1 (Prosumer) *We define a prosumer* Pros *as a set of power functions.*

[1] Instances are essentially runs of the system in question

Given a power function $P \in \text{Pros}$, we define the displaced energy of Pros in P over a time interval $[t_1, t_2]$ as $\Delta E = \int_{t_1}^{t_2} P(t)dt$ and refer to it as the *prosumed energy* over that interval (see Fig. 2.1). The aggregation of multiple prosumers in a network leads to adding up their power functions.

Definition 2 (Aggregation of prosumers) *For power functions P_1 and P_2 we define $(P_1 + P_2)(t) = P_1(t) + P_2(t)$ for all t. For prosumers Pros_1 and Pros_2 we define $\text{Pros}_1 + \text{Pros}_2 = \{(P_1 + P_2)(t) \mid P_1 \in \text{Pros}_1, P_2 \in \text{Pros}_2\}$.*

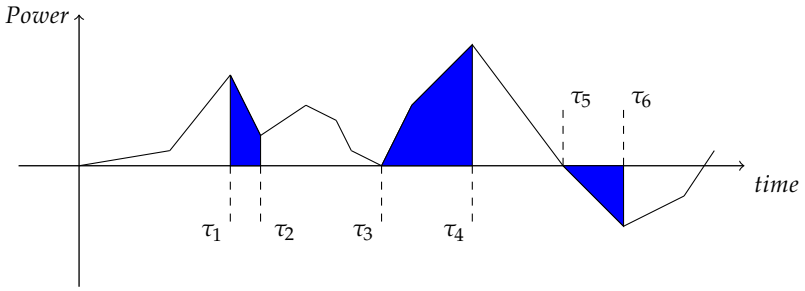


Figure 2.1: An example power function $P(\tau)$ defining the rate of prosumption in time. The blue areas describe the energy prosumed in time intervals $[\tau_1, \tau_2]$, $[\tau_3, \tau_4]$, and $[\tau_5, \tau_6]$.

2.1 Flexoffers

Now that we have formally defined Resource systems and established an approach to reason about prosumer behavior in terms of resource flow as power, we now turn to identifying the desired flexibility properties. In previous literature, the property *flex-offer* has been introduced to capture power behavior in time. Flex-offers define time dependent energy bounds for which a prosumer is willing to adhere. Since we have characterized resource behavior as flow over time it is straightforward to define flexoffers in this setting as energy difference over time. Intuitively, prosumers define power behavior and flexoffers capture the integral of power over time i.e. energy, referred to as prosumption.

Flexibility analysis of prosumers consists of determining whether or not a prosumer can adhere to a flexoffer, which consists of a set of time dependent prosumption bounds, called energy slices. The energy slices make up a sequential pattern of energy bounds that a prosumer is ready to commit to. (see Fig. 2.2).

Definition 3 (Energy Slices and Flexoffers) *An energy slice $s = ([t_l, t_u], [e_l, e_u])$ is a tuple defining an observation window $[t_l, t_u]$ and a bound on the energy prosumption $[e_l, e_u]$. A flexoffer $F = \{s_1, s_2, \cdots s_n\}$ consists of a set of n (disjoint) energy slices. Formally, we say that a power function P complies with F, denoted $P \models F$ if: $\forall([t_l, t_u], [e_l, e_u]) \in F : \int_{t_l}^{t_u} P(\tau)\, d\tau \in [e_l, e_u]$.*

Definition 4 (Prosumption equivalence) *We say that power functions P_1 and P_2 exert equivalent prosumption behavior with respect to a flexoffer F, denoted $P_1 =_F P_2$ if: $\forall([t_l, t_u], \bullet) \in F : \int_{t_l}^{t_u} P_1(t)\, dt = \int_{t_l}^{t_u} P_2(t)\, dt$.*

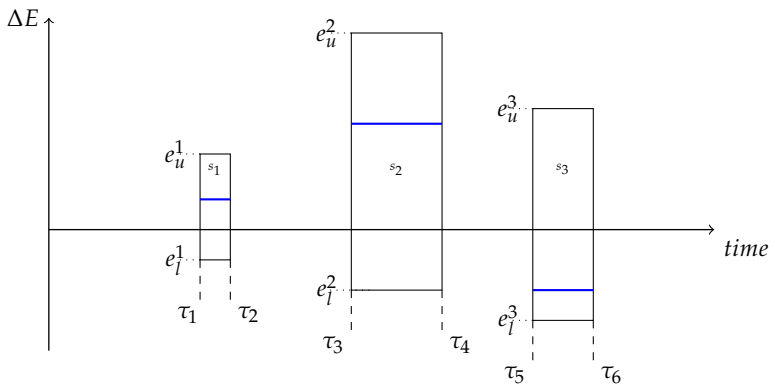


Figure 2.2: An example flexoffer $F = \{s_1, s_2, s_3\}$ s.t. $s_i = ([t_l^i, t_u^i], [e_l^i, e_u^i])$ for all $i \in \{1, 2, 3\}$. Energy slices are defined over intervals $[\tau_1, \tau_2]$, $[\tau_3, \tau_4]$ and $[\tau_5, \tau_6]$. Here times are slice specific lower or upper time bounds, e.g. $[\tau_1, \tau_2] = [t_l^1, t_u^1]$. The blue lines correspond to the prosumption of P defined in Fig 2.1. Clearly $P \models^{\exists} F$.

Given a flexoffer F, we can restrict the behavior of a prosumer Pros to those power functions $P \in$ Pros that comply with F. If this restriction is non-empty, we say that F is *feasible* in Pros, denoted Pros $\models^{\exists} F$ (see Fig. 2.2).

Definition 5 (Flexoffer Feasibility) *Let F be a flexoffer and* Pros *a prosumer. We define feasibility of F in* Pros *as follows:* Pros $\models^{\exists} F$ *iff* $\exists P \in$ Pros $: P \models F$

Alternatively, if the prosumer contains every possible power function that complies with F, we say that Pros completely satisfies F, denoted Pros $\models^{\forall} F$.

Definition 6 (Flexoffer Satisfiability) *Let F be a flexoffer and* Pros *a prosumer. We define complete satisfaction of F in* Pros *as follows:* Pros $\models^{\forall} F$ *iff* $\forall P \models F \, \exists P' \in$ Pros $: P' =_F P$.

In addition to prosumption flexibility, in the original definition in [11], flexoffers encompasses a notion of start time flexibility, capturing the capacity of a prosumer to wait before starting. For simplicity, we omit this aspect, but do note that we could extend the method presented in this paper to handle this as well. In the following we focus on flexoffers for which the requested start time has already been chosen.

Now that we have all the preliminary definitions in place, the balancing problem in terms of flexoffers can be addressed by either directly analyzing an entire aggregated prosumer or by contract negotiations between individual prosumers. A direct analysis comes down to determining whether a zero-flexoffer is feasible in an aggregated prosumer Pros, i.e. Pros $\models^{\exists} F^0$, where F^0 denotes a flexoffer in some time interval for which all slices define 0-energy prosumption bounds (or something close to 0 in a more loose analysis).

In the indirect approach, prosumers negotiate flexoffers to achieve balance. A prosumer (or aggregation of prosumers) that offers to completely satisfy a flexoffer F has to be balanced by a prosumer (or aggregation) for which the negated flexoffer $-F$ is feasible.

Definition 7 (Negated Flexoffer) *Let F be a flexoffer, then by $-F$ we denote the negation of F defined as follows:* $-F = \{([t_l, t_u], [e'_l, e'_u]) \mid ([t_l, t_u], [e_l, e_u]) \in F \wedge e'_l = -e_l \wedge e'_u = -e_u\}$

Theorem 1 (Balance) *For prosumers A and B and flexoffer F, if $A \models^{\forall} F$ and $B \models^{\exists} -F$ then for every $P \in B$ with $P \models -F$ there exists a $P' \in A$ such that $P + P' =_F 0$. We say that A and B are balanced.*

This theorem follows directly from the definition of satisfiability and negation. Note that, following the scheme of Theorem 1 in practice means that Prosumer B only promises to adhere to flexoffer $-F$ in some way, while prosumer A promises to be follow along with any choice of P'.

Chapter 3

Resource Timed Automata

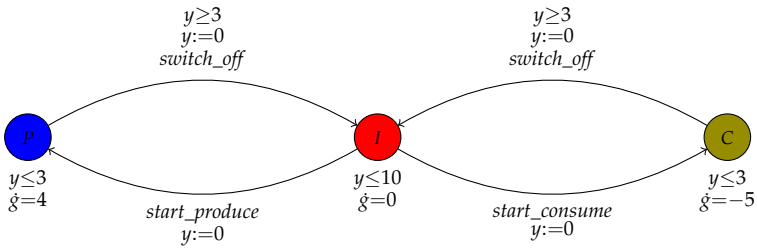


Figure 3.1: RTA Pros defined over clock y and resource g. Location P describes a mode of production where delays result in a positive rate of change of 4 in resource g. Location C describes a mode of consumption where there is a negative rate of change of 5 in g. Location I describes an idle mode, in which the rate of change in g is 0. The system resides in location P for 3 time units, after which it makes a transition to mode I. Similarly, in mode C it resides for 3 time units. Mode I can be left immediately, but must be left within 10 time units. After each transition the clock is reset to 0.

To model prosumers we develop a suitable automata formalism called *Resource Timed Automata*. Extending the work on bounded infinite behavior in Energy Timed Automata (ETA) done by Bacci et. al. in [2], we study the multi variable setting of finite behavior in Resource Timed Automata (RTA). RTAs extends ETAs[2] in the sense that they generalize the concept of a single energy variable into a set of resource variables. In each state of the their execution, RTAs defines the flow of its associated resources as constant derivatives in time. As such the resource variables themselves captures the integrated flow over time. RTA's are a strict subclass of Hybrid Automata [7] since the behavior of the automaton does not depend on the value of the resources. In particular, there are no resource guards on the transitions and no resource constraints in the states. RTA's are reminiscent of Weighted Timed Automata, with a crucial difference in the way in which parallel composition is defined.

RTAs are defined over a set of real valued clocks $\mathcal{C}$. A clock stores the amount of time elapsed since last reset, captured by a valuation $\varsigma : \mathcal{C} \to \mathbb{R}_{\geq 0}$.
Let $d \in \mathbb{R}_{\geq 0}$, $x \in \mathcal{C}$ then $\varsigma + d$ is defined as: $(\varsigma + d)(x) = \varsigma(x) + d$

Resetting a set of clocks $r \subseteq \mathcal{C}$, captured by $\varsigma[r]$ is defined as: $\varsigma[r](x) = \begin{cases} 0, & x \in r \\ \varsigma(x), & otherwise \end{cases}$

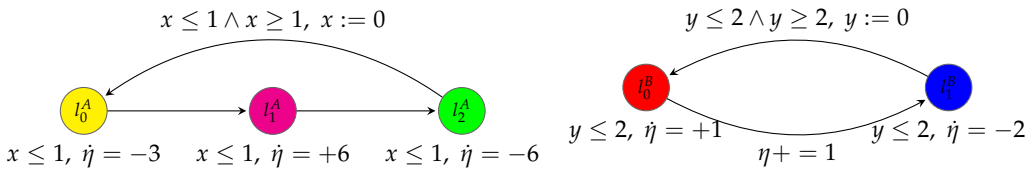
Additionally, clocks can be numerically tested in constraints.

Definition 8 (Clock constraint and evaluation) *Let $\mathcal{C}$ denote a set of clocks. We define the set $\mathcal{B}(\mathcal{C})$ of clock constraints over $\mathcal{C}$, by the abstract syntax: $g := x \sim c \mid g \wedge g$ where $x \in \mathcal{C}$, $c \in \mathbb{Q}_{\geq 0}$ and $\sim \in \{\leq, \geq\}$.*
We define the evaluation of a clock constraint in some valuation $\varsigma \models g$ inductively on the structure of g as follows: $\varsigma \models x \sim c \Leftrightarrow \varsigma(x) \sim c$ and $\varsigma \models g_1 \wedge g_2 \Leftrightarrow \varsigma \models g_1 \wedge \varsigma \models g_2$, where $g_1, g_2 \in \mathcal{B}(C)$.

We denote the initial clock valuation as ς_0. Let $x \in \mathcal{C}$ then $\varsigma_0(x) = 0$.

RTAs are also defined over a set of real valued variables $\mathcal{E} = \{\eta_1, \eta_2, \cdots, \eta_n\}$ called resources. These are captured by a valuation $\rho : \mathcal{E} \to \mathbb{R}$.
For $f : \mathcal{E} \to \mathbb{Q}$, $\eta \in \mathcal{E}$, and $d \in \mathbb{R}_{\geq 0}$, we define the two following operations: $(\rho + f)(\eta) = \rho(\eta) + f(\eta)$ and $(f \cdot d)(\eta) = f(\eta) \cdot d$.

To reason about observable behavior, RTAs are defined over a set of actions $\text{Act} = \Sigma \cup \epsilon$, where Σ is some alphabet and ϵ denotes the empty action.



(a) RTA A defined over clock x and resource η

(b) RTA B defined over clock y and resource η

Figure 3.2: RTAs A and B defined over resource η. Update rates are defined for η in each location. lack of discrete updates of edges, simply denotes an update of 0. Note that no actions are related to the edges of A or B. Whenever the action defined over an edge is the empty action ϵ, we simply omit it from depictions.

Definition 9 (Resource Timed Automata) *We define an RTA $(L, L_0, E, \text{rate}, \text{inv}, \text{Act}, \mathcal{C}, \mathcal{E})$ as a tuple consisting of:*

- *finite sets of clock variables $\mathcal{C}$, resource variables $\mathcal{E}$, and actions* Act*;*
- *a finite set of locations L, and a non-empty subset $L_0 \subseteq L$ of initial locations;*
- *a finite set of transitions $E \subseteq L \times \mathcal{B}(\mathcal{C}) \times \text{Act} \times 2^{\mathcal{C}} \times (\mathcal{E} \to \mathbb{Q}) \times L$;*
- *a location-rate function* $\text{rate} : L \to (\mathcal{E} \to \mathbb{Q})$*;*
- *a location-invariant function* $\text{inv} : L \to \mathcal{B}(\mathcal{C})$.

If $a \in \text{Act}$ and $(l, g, a, r, i, l') \in E$ we say that $(l \xrightarrow{g,a,r,i} l')$. Figure 3.2 depicts two small example RTAs.

When depicting an RTA as in Fig. 3.1, an update rate $\text{rate}(l)(e) = x$ of resource e in location l will be annotated by writing $\dot{e} = x$ below or above the location l, except when the rate is 0 in which case it is left out completely. Similarly, a transition $(l, g, a, r, i, l') \in E$

is depicted as an arrow from l to l' annotated with a and g. If for some clock variable $c \in \mathcal{C}$ we have a reset (i.e. $c \in r$), this is annotated on the transition by writing $c := 0$. Similarly, for a resource variable $e \in \mathcal{E}$ that has an increment or decrement $i(e) = x$, this is annotated by $e+ = x$, unless $x = 0$ in which case the annotation is left out.

Definition 10 (Semantics of RTAs) *Let $A = (L, L_0, E, \text{rate}, \text{inv}, \text{Act}, \mathcal{C}, \mathcal{E})$ be an RTA. The Resource Timed Labeled Transition System (RTLTS) generated by A is defined as: $T = (S, Lab, \omega, \longrightarrow)$,*
where:

- *S is a set of states (configurations) defined as $S = \{(l, \varsigma, \rho, \tau) \mid (l, \varsigma, \rho, \tau) \in L \times (\mathcal{C} \to \mathbb{R}_{\geq 0}) \times (\mathcal{E} \to \mathbb{R}) \times \mathbb{R}_{\geq 0} \wedge \varsigma \models \text{inv}(l)\}$*
- *Lab is a set of labels defined as $Lab = \text{Act} \cup \mathbb{R}_{\geq 0}$*
- *$\omega : S \to (\mathcal{E} \to \mathbb{R})$ defines a mapping from states to resource valuations. If $s = (l, \varsigma, \rho) \in S$ and $\eta \in \mathcal{E}$ we have $s(\eta) = \omega(s)(\eta) = \rho(\eta)$*
- *$\longrightarrow$ is a transition relation $\longrightarrow = \{\xrightarrow{\alpha} \mid \alpha \in Lab\}$ defined as:*
  - $(l, \varsigma, \rho, \tau) \xrightarrow{a} (l', \varsigma', \rho', \tau)$
    if there is an edge $(l \xrightarrow{g,a,r,i} l')$ s.t. $\varsigma \models \text{inv}(l)$, $\varsigma \models g$, $\rho' = \rho[r]$, $\rho' \models \text{inv}(l')$, $a \in \text{Act}$ and $\rho' = \rho + i$
  - $(l, \varsigma, \rho, \tau) \xrightarrow{d} (l, \varsigma + d, \rho', \tau')$
    s.t. $d \in \mathbb{R}_{\geq 0}$, $\forall d' \in [0, d]$. $\varsigma + d' \models \text{inv}(l)$, $\tau' = \tau + d$ and $\rho' = \rho + \text{rate}(l) \cdot d$

If $l_0 \in L_0$, $\varsigma_0 \models \text{inv}(l_0)$ and $\rho : \mathcal{E} \to \mathbb{R}$, then $(l_0, \varsigma_0, \rho, 0)$ denotes an initial state of T.

Consider RTA A depicted in figure 3.2. Let $s_0 = (l_0^A, \varsigma_0, \rho, 0)$, where $s_0(\eta) = 3$ be an initial state of A. Let T_a be the RTLTS generated by A.
The transition relation of T_a defines infinitely many delay transitions from s_0 eg. $(l_0^A, x : 0, \eta : 3, 0) \xrightarrow{0.5} (l_0^A, x : 0.5, \eta : 1.5, 0.5)$. It also defines a discrete transition from s_0: $(l_0^A, x : 0, \eta : 3, 0). \xrightarrow{\epsilon} (l_1^A, x : 0, \eta : 3, 0)$.

Semantically, an RTA as depicted in fig 3.1 represents a set of runs, i.e. possible evolutions of states and resources over time, which are defined as follows.

Definition 11 (Runs of an RTA) *Given an RTA defined as $(L, L_0, E, \text{rate}, \text{inv}, \text{Act}, \mathcal{C}, \mathcal{E})$, a* run *$(x_0, t_0, a_0, x_1, t_1, a_1, ..., x_N, t_N)$ of length N defines an alternating sequence of states x_n, delays $t_n \in \mathbb{R}_{\geq 0}$, and actions $a_n \in$ Act such that the run defines a path in the RTLTS generated by the RTA.*

The run is said to reach *a state $y = (l, \varsigma, \rho, \tau)$ at some time $s \in \mathbb{R}_{\geq 0}$, and y is hence called* reachable, *whenever $\tau = s$ and there exists an $n \leq N$ such that $x_n = (l, \varsigma', \rho', \tau')$ with $\tau' \leq s \leq \tau' + t_n$ and $\varsigma = \varsigma' + (s - \tau')$, and $\rho(e) = \rho'(e) + \text{rate}(l)(e) \cdot (s - \tau')$ for all $e \in \mathcal{E}$.*

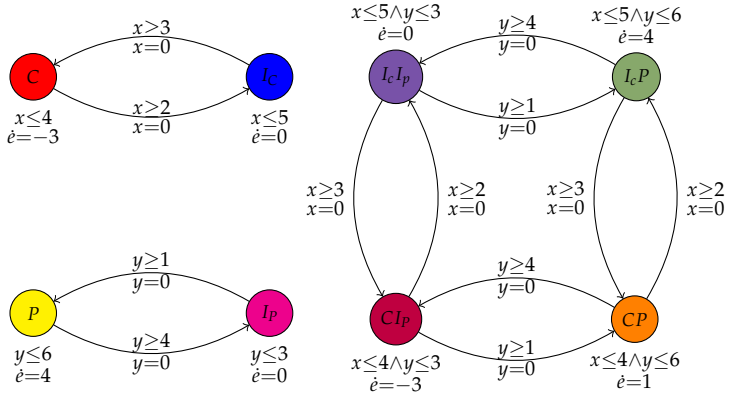


Figure 3.3: On the left we see two RTAs defined over resource variable e and clock x respectively y. On the right we see the parallel composition of the two RTAs defined as the product automata of the two. The red-blue RTA models a simple consumer. It can remain idle between 3 and 5 time units after which it can consume with rate -3 between 2 and 4 time units. The yellow-magenta RTA models a simple producer which can idle between 1 and 3 time units before switching to producing with rate 4 between 4 and 6 time units. The parallel composition of the two defines a simple prosumer which can exert prosumption behavior defined by the sum of production and consumption.

Definition 12 (Network of RTAs) *For $1 \leq i \leq m$, let $A_i = (L_i, L_0^i, E_i, \text{rate}_i, \text{inv}_i, \text{Act}_i, \mathcal{C}_i, \mathcal{E})$ be an RTA.*
The network A is defined as the parallel composition:

$$A = A_1 \otimes A_2 \otimes \cdots \otimes A_m = (\mathcal{L}, \mathcal{L}_0, E', \text{rate}', \text{inv}', \text{Act}', \mathcal{C}', \mathcal{E}')$$

where:

- $\mathcal{L} = L_1 \times L_2 \times \cdots \times L_m$
- $\mathcal{L}_0 = L_0^1 \times L_0^2 \times \cdots \times L_0^m$
- $E' = \{(\{l_1, l_2, \cdots l_j, \cdots l_m\}, g, a, r, i, \{l_1, l_2, \cdots l'_j, \cdots l_m\}) \mid \exists j \in [1, m] : (l_j, g, a, r, i, l'_j) \in E_j\}$
- $\text{rate}' = \bigcup_{i=1}^m \text{rate}_i$
- $\text{inv}' = \bigwedge_{i=1}^m \text{inv}_i$
- $\text{Act}' = \bigcup_{i=1}^m \text{Act}_i$
- $\mathcal{C}' = \bigcup_{i=1}^m \mathcal{C}_i$

st. $\bigcap_{i=1}^m \mathcal{C}_i = \emptyset$

Fig 3.3 illustrates an example of two small RTA's in aggregation. Furthermore, in fig 3.4 we visualize the valuation of the resource variable e throughout one of the runs of that parallel composition.

Definition 13 (Semantics of Networks of RTAs) *Let* $A = A_1 \otimes A_2 \otimes \cdots \otimes A_m = (\mathcal{L}, \mathcal{L}_0, E', \text{rate}', \text{inv}', \text{Act}', \mathcal{C}', \mathcal{E}')$ *be a network of RTAs. The RTLTS generated by A is defined as:* $T = (S, Lab, \omega, \longrightarrow)$*, where*

- *S is a set of states defined as* $S = \{(\mathcal{L}, \varsigma, \rho, \tau) \mid L_1 \times L_2 \times \cdots \times L_m \times (C \to \mathbb{R}_{\geq 0}) \times (\mathcal{E} \to \mathbb{R}) \times \mathbb{R}_{\geq 0} \wedge v \models I(\mathcal{L}) \wedge \varsigma \models \text{inv}(\mathcal{L})\}$
- *Lab is a set of labels defined as* $Lab = Act \cup \mathbb{R}_{\geq 0}$
- $\omega : S \to (\mathcal{E} \to \mathbb{R})$ *defines a mapping from states to resource valuations. If* $s = (l, \varsigma, \rho, \tau) \in S$ *and* $\eta \in \mathcal{E}$ *we have* $s(\eta) = \omega(s)(\eta) = \rho(\eta)$
- $\longrightarrow$ *is a transition relation* $\to = \{\xrightarrow{\alpha} \mid \alpha \in Lab\}$ *defined as:*
  - $(\mathcal{L}, \varsigma, \rho, \tau) \xrightarrow{a} (\mathcal{L}', \varsigma', \rho', \tau)$
    if $a \in \text{Act}_i$ *and there is an edge* $(l_i \xrightarrow{g,a,r,i} l_{i'})$*, then* $\varsigma \models g, \varsigma \models \text{inv}'(\mathcal{L}), \varsigma' = \varsigma[r]$*,* $\varsigma' \models \text{inv}'(\mathcal{L}')$ *and* $\rho' = \rho + i$
  - $(\mathcal{L}, \varsigma, \rho, \tau) \xrightarrow{d} (\mathcal{L}, \varsigma + d, \rho', \tau')$
    s.t. $d \in \mathbb{R}_{\geq 0}$*,* $\forall d' \in [0, d]. \varsigma + d' \models \text{inv}'(\mathcal{L})$*,* $\tau' = \tau + d$ *and* $\rho' = \rho + \text{rate}'(\mathcal{L}) \cdot d$*, where* rate′ *is defined as:* $\text{rate}'(\mathcal{L}) = \sum_{j.\eta \in \mathcal{E}_j} \text{rate}_j(l_j)$

Let $\mathcal{L}_0 \in L_0^1 \times L_0^2 \times \cdots \times L_0^m$, if $v_0 \models \mathcal{L}_0$ and $w : \mathcal{E} \to \mathbb{R}$, then $(\mathcal{L}_0, \varsigma_0, \rho, 0)$ denotes an initial state of T.

We now show how the power function exerted by prosumers modeled as RTAs can be captured through runs. This links the original flexibility analysis to the problem of modelchecking RTAs in general.

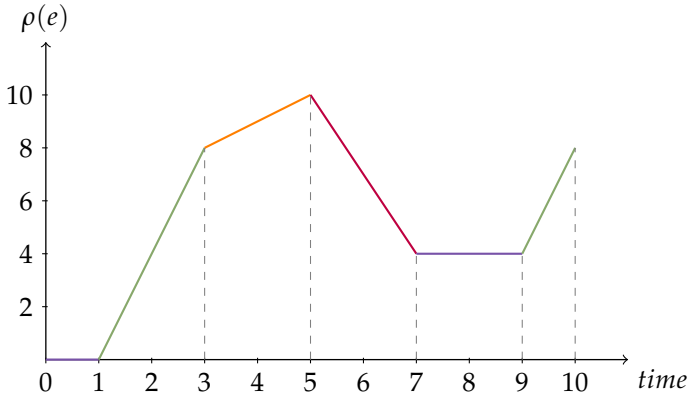


Figure 3.4: An arbitrary run of the parallel composition depicted in Fig 3.3. Time intervals defined by the distance between dashed lines denotes delays in locations: $(I_C I_P)$, $(I_C P)$, (CP), $(C I_P)$, $(I_C I_P)$.

Resource Timed Automata as in definition 9 can straightforwardly be used to model prosumers as in definition 1. After all, let e be a resource variable of an RTA representing energy, then we can associate any given run $(x_0, t_0, a_0, \ldots, x_N, t_N)$ of that RTA with an energy function E such that, if the run reaches state $y = (l, \varsigma, \rho, \tau)$ at time $t = \tau$, then $E(t) = \rho(e)$. If the transitions of the RTA do not contain instantaneous changes to e, this

function E is differentiable and its derivative $P(t) = \dot{E}(t)$ is a piece-wise constant power function, and aggregation of prosumers according to definition 2 and parallel composition of their RTA models according to definition 12 coincide as follows.

Theorem 2 *Given two RTA's X and X' defined over a resource variable e, if X has a run with associated power function P and X' has a run with associated power function P', then $X \otimes X'$ has a run with associated power function $P + P'$.*

If the transitions of the RTA do contain instantaneous changes, the notion of flexibility analysis still makes sense for runs, because it only checks the prosumed energy and does not rely on differentiability as such.

Chapter 4

Flexibility Analysis using Uppaal

Our final step in performing flexibility analysis is to discretize the RTAs that model our prosumers and make their parallel composition suitable for analysis in the Uppaal modelchecker [5]. First, we give a discretization procedure from RTAs to Timed Automata, and finally we discuss how to capture a flexoffer as a discrete observer automaton. This reduces flexibility analysis to reachability problems that can then be efficiently solved using Uppaal.

Given a parallel composition of RTAs as a model of an aggregation of prosumers, we are left with the problem of modelchecking this model. Our tool of preference is Uppaal, because its approach to reachability analysis using randomized depth-first search is expected to scale well for the type of problem that we are studying in this project. Uppaal, however, is not able to check continuous system dynamics. Therefore we adapt a known discretization technique[9], to turn a parallel composition of RTAs into a parallel composition of Timed Automata extended with integer variables and hybrid clocks[1] that can be analyzed using Uppaal. We use primarily Uppaal Classic in order to conduct verification on the discretized prosumers. To simulate runs we use Uppaal SMC which gives us the option to simulate sample runs under certain constraints at a known time horizon.

The basis of the discretization, is that we restrict the runs of our RTA to those in which discrete timesteps are made. This will lead to a subset of the original set of runs of the RTA, but if a schedule is found in the discretized automaton we are certain it also exists in the original RTA. The first step in the discretization is to introduce, for every resource variable e in the RTA, two discrete variables E and RateE - keeping track of the resource and its rate - together with an update automaton (see Fig. 4.1) that updates E using the value of RateE whenever one time unit has passed in the Timed Automaton model ($E := E + \text{RateE}$). The second step is to remove the continuous behavior of e from the RTA, and replace it with communication to the update automaton, adjusting the value of RateE and updating E with possible instantaneous changes. This removal results in a timed automaton (see Fig. 4.2), and placing multiple of such timed automata in parallel with a single copy of each update automaton provides a discretization of the parallel composition of RTAs as a whole. Lastly, whenever RateE is updated we must insure that a pending update to E has

[1] These clocks are not part of the discretization technique but rather helps give insights on sample runs of the system

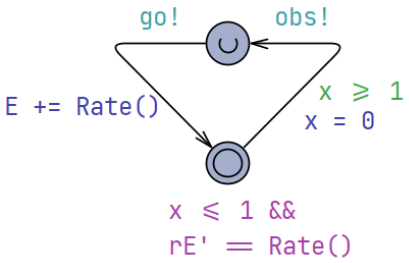


Figure 4.1: An update automaton for resource e consists of two locations which updates the discrete variable E with the rate Rate() (a function that returns the accumulated rate of e) whenever 1 time unit has passed. the variable rE is a so-called hybrid clock, describing the real evolution of e over time. Synchronization on channel obs insures correctly timed observation of rates in the FlexObserver automaton. synchronization on channel go insures that E is updated before any updates to $RateE$. The location at the top is so-called urgent insuring that no delays can occur in this location.

already occurred. We therefore introduce so-called broadcast synchronization on channel go making all transitions that update $RateE$ input enabled on said channel. Whenever the update automaton is altering E it communicates to all components that it needs to make the update before $RateE$ is altered anywhere. For convenience, instead of having one $RateE$ for each resource variable, for M prosumers we instead define the array structure $RateE[M]$ for each resource variable. Here $RateE[id]$ defines the prosumer specific rate of a given resource in order to more easily distinguish between them. Furthermore, we define function $Rate()$ which simply evaluates the sum of all $RateE[id]$.

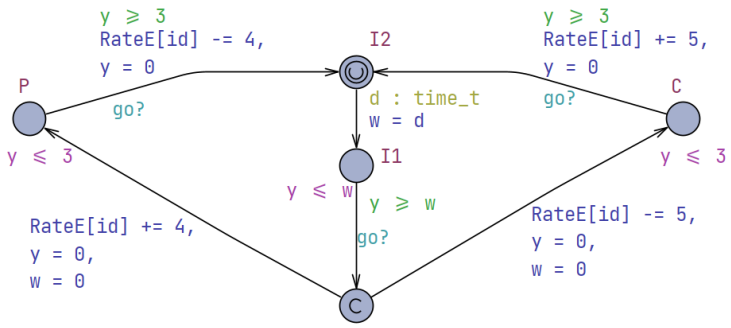


Figure 4.2: The discretized posumer automaton consists of a copy of the RTA prosumer automaton from which references to resource variables e have been removed and to which communications updating the rate $RateE[id]$ have been added. Here id is the prosumer id. Note that the updates to RateE[id] on the transitions reflect *changes* in the rate rather than the actual rate specified in the corresponding locations of the original RTA in Fig. 3.1. The edge from location I_2 to I_1 defines a so-called select statement on type time_t, a bound integer type in the range $[0, 10]$. this edge defines 11 different edges, one for each integer value in the range. This captures the original idle location's choice of delay described in Fig. 3.1.

As a final step, in order to actually conduct analysis on a given flexoffer, we model the flexoffer as an observer automaton (see Fig. 4.3) which is placed in parallel with the discretized RTAs and update Automata. We exploit the syntactic features of Uppaal for this, and encode a flexoffer and its associated slices into appropriate data structures: *slice_t*

and *flex_t*. The type *slice_t* defines a struct of four integer variables t_1, t_2, e_l and e_u and the type *flex_t* defines a struct of one array of slices called *slices* of size S, mimicking definition 3. The FlexObserver automaton waits for the relative time in which the next slice will occur. After entering slice i, the observer keeps track of the prosumption DeltaE occurring during the slice. If *DeltaE* violates the prosumption bound specified by the slice after duration *dur*, the observer moves to location NotSatisfied and deadlocks. Upon reaching the end of the last slice, if no violation has occurred thus far the observer moves to location Satisfied, indicating that the flexoffer is indeed feasible for that particular execution.

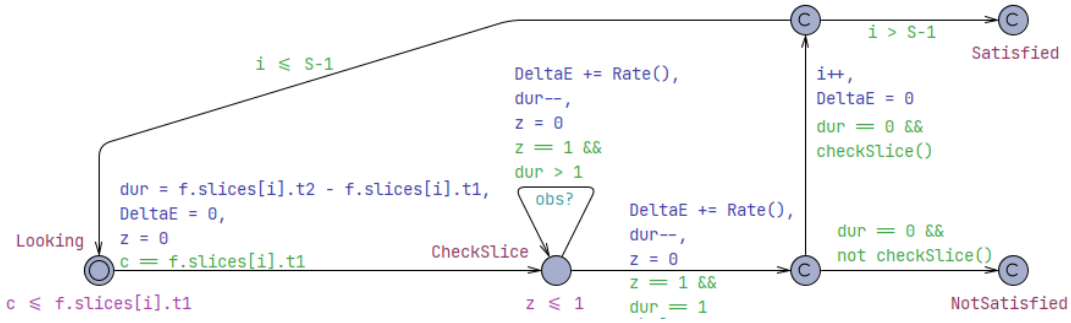


Figure 4.3: The *FlexObserver* automaton. When observation concludes it will be deadlocked in location *NotSatisfied* or *Satisfied*. Some locations are so-called *committed* (indicated by the letter *C*) which means that time cannot pass in such a location and the next transition must include such locations. In this case it renders the parallel composition deadlocked.

We can now verify whether a flexoffer is feasible using Uppaal by querying simply if there exists a run in which the location Satisfied is reached. Furthermore we can ask to generate such a run, which provides us immediately with a viable schedule witnessing that the flexoffer is feasible in the prosumer aggregation. Satisfiability of a flexoffer can be verified by querying for each discrete run in the interval defined by the flexoffer whether location Satisfied is reachable. In the current model, the quantification of runs must be inputted manually through variable f however this could easily be automated as extensions of Uppaal itself. Naturally these quantifications only works because we are dealing with the discrete domain and therefore are guaranteed to have finitely many distinct runs captured by a given flexoffer. As an illustration, in Fig. 4.4 we show a run as it was generated for a parallel composition of 4 prosumers as defined in Fig. 4.2. Individual schedules for the prosumers were generated and can be inspected using the concrete simulator option in Uppaal, but are left out of the illustration.

We now conduct a small sample flexibility analysis on said four prosumer composition, which we refer to as Pros. Let $F = \{([1,2],[0,0]),([3,5],[-20,-21]),([7,9],[11,12])\}$ be a flexoffer. Observe that flexoffer F describes perfect balance in time interval $[1,2]$ a relatively large amount of consumption in $[3,5]$ and a notable production excess in $[7,9]$. We can check whether F is feasible in *Pros*: $Pros \models^{\exists} F$ by asking Uppaal whether there exists a run of *Pros* where location *Satisfied* is reached. This property turns out to be true and therefore $Pros \models^{\exists} F$ is also true. To check whether *Pros* satisfies F: $Pros \models^{\forall} F$ we verify whether location *Satisfied* is reachable for some run in *Pros* for each possible combination of prosumption values in each slice. In Uppaal we simply verify the property for each of the following flexoffers:

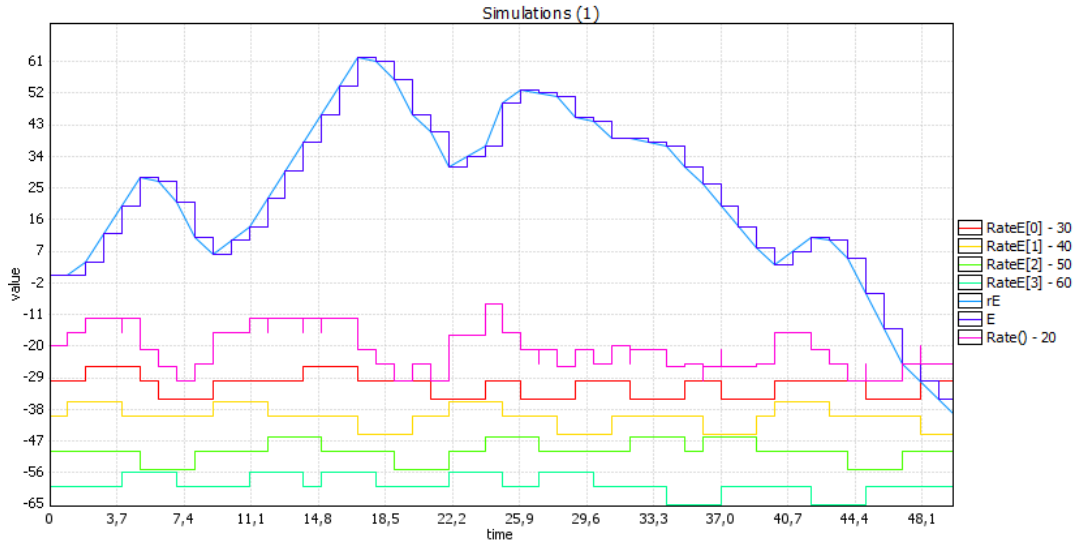


Figure 4.4: A sample run of the parallel composition of four discretized prosumers as depicted in Fig. 4.2. rE and E describes the real respectively discrete prosumed energy. For $0 \leq i \leq 3$, $RateE[i]$ describes the individual rate of prosumer i with $Rate()$ describing the accumulated rate. Here the rates are depicted with an offset for convenience.

- $\{([1,2],[0,0]),([3,5],[-20,-20]),([7,9],[11,11])\}$,
- $\{([1,2],[0,0]),([3,5],[-20,-20]),([7,9],[12,12])\}$,
- $\{([1,2],[0,0]),([3,5],[-21,-21]),([7,9],[11,11])\}$,
- $\{([1,2],[0,0]),([3,5],[-21,-21]),([7,9],[12,12])\}$

This property turns out to be true and therefore $Pros \models^{\forall} F$ is also true. Initial experiments in Uppaal show that up to 100 simple prosumer models like the one depicted in Fig. 4.2 can still be verified within 10 seconds, so the approach seems to scale well in practice for the kind of modelchecking problems we are studying.

The reason for this, is that the randomized depth first search as implemented in Uppaal is generally fast for problems in which a correct run exists and only one valid run needs to be generated. Naturally, if a correct run does not exist, the entire statespace needs to be generated, which grows exponentially with the number of prosumers, resources and clocks.

Chapter 5

Conclusion

5.1 Conclusion

We have shown how to use Resource Timed Automata to model prosumer behavior, and how to interpret flexibility analysis as reachability problems on a parallel composition of such automata. We have furthermore shown how to translate these automata into Timed Automata extended with integer variables and hybrid clocks by adapting a standard discretization techniques for use with the resource-additive parallel composition defined for RTAs. Initial results on the use of Uppaal Stratego for finding prosumer schedules that satisfy a given load balancing condition are hopeful. Nevertheless, a more realistic case study still needs to be explored in order to confirm that the schedulability problem indeed scales well under randomized depth-first search whenever a solution exists.

We note that, in principle, RTAs are a subclass of hybrid automata, so they could be studied using modelchecking tools that are specific for this extended formalism as well. This would avoid the necessity of discretization, but to the best of our knowledge there are no tools available yet that support randomized depth-first search for this extension. Furthermore, we suspect that RTAs have decidability advantages that do not hold for hybrid automata in general, but this is left as a topic for future research.

5.2 Future Work

In this last section we outline ongoing projects at the time of writing. A general point of note is that the prosumer flexoffer analogue defined in chapter 2 is an instance of a larger abstraction property. In this report, the restricted version has been outlined because that was all that was needed for the technique in question. However, a more powerful general abstraction has been developed of which the previously defined is simply a special case of.

As mentioned earlier, we use the classic implementation of Uppaal, however the TIGA and STRATEGO version have also been used for some of the projects listed below. Through TIGA we get tools for synthesizing controllers for more complex abstractions such as Timed Games. This is useful when considering Resource Systems that are subject to some optimization such as safe ranges in resource variables. Uppaal Stratego presents means for

conducting Statistical learning on Stochastic Priced Timed Games which again expands the on which types of statistical optimization problems we can conduct on Resource Systems.

5.2.1 Exact Analysis

The biggest shortcoming of the discretization method described in this report is the inability to exactly capture the behavior of prosumers. Another method has therefore been developed based on solving linear arithmetic expressions. Because of the various restrictions enforced on RTAs distinguishing them from Linear Hybrid Automata we are guaranteed that any RTA can be described as a finite number of linear inequalities. Such linear systems of inequalities can be solved very efficiently using methods such as quantifier elimination. An ongoing project aims at exploiting these properties in order to generate easily analyzable system abstractions.

5.2.2 Energy Contracts

Exploring the balancing property as defined in definition 1 serves as the foundation for casting energy based assume-guarantee properties into this restricted hybrid domain of which RTAs are defined. Enriching an existing formalism such as I/O-Timed Automata with resource information gives insight into scheduling properties and abstraction problems. It is strongly suspected that established abstraction methods and operations such as the quotient operation in this setting could help solving the balancing problem for large scale resource systems.

5.2.3 Bounded Infinite Runs

From a theoretical point of view the RTA formalism is interesting because it expands on the already established framework of ETA. Investigating how the multi variable setting affects state of the art techniques and decidability results could help identifying an exact frontier of decidability. Of particular note is the bounded infinite run problem in the multi variable setting. This problem has already been shown to be decidable for a realistic restricted version of ETAs. A RTA analogue of this result would be a significant contribution.

5.2.4 Regularity

Based on the characterization briefly described in Chapter 2 there is evidently some grounds for developing a Resource analogue Kleene theorem. Such a theorem could help in identifying frontiers of decidability and also serve as mathematical foundation for time bound reachability. This project is in its infancy at the time of writing but it is suspected that the method developed in project 5.2.1 gives rise to a general framework based on a resourced timed language.

The main idea is to first cast the notion of *Resource System* into an algebraic framework in the same manner as in [1].

Semantically, Resource Systems defines sequential discrete behavior enriched with timing and resource information. The goal is to capture both event based and state based resource behavior, which can be accomplished by characterizing it as a mix of an event sequence and signals in which timing is also captured as signals. In this context, signals

are viewed as functions from the non-negative real line $\mathbb{R}_{\geq 0}$ to some discrete observation alphabet Σ.

We define this mix of event-sequence and signals as the free shuffle under an appropriate reduction congruence of certain monoids, each representing a different formal aspect of the behavior; time, resource and discrete actions. First we define the semi-ring $\mathcal{W} = (W, \oplus, \otimes, \mathbf{0}, \mathbb{1})$ which we will refer to as the resource semi-ring (the same structure defined in 2). The monoid $\mathcal{W}_{\oplus}$ over $\mathcal{W}$ generated by $\oplus$ defines the resource arithmetic. We define $\mathfrak{N}_{+}$ as the monoid over the hybrid timeline $\mathbb{R} \times \mathbb{N}$ generated by $+$ (arithmetic addition), which defines both discrete and continuous resource behavior. Lastly the free monoid Σ^* defines discrete observations.

Let $\mathcal{W}$ be an m-element set and $\{\mathfrak{N}_{+_w} \mid w \in \mathcal{W}\}$ be m distinct copies of $\mathfrak{N}_{+}$. Then we define the Timed resource signal-event monoid defined as: $\mathcal{F} = \coprod_{w \in \mathcal{W}} \mathfrak{N}_{+_w} \amalg \Sigma^*$ where $\amalg$ defines the free shuffle under a reduction congruence. A typical element of $\mathcal{F}$ looks like:

$$2.5_a * y * 5.7_b * c * x * 1.3_d$$

where $a, b, c, d \in W$ and $x, y \in \Sigma$. Delays are annotated with a rate from W and discrete resource updates happens before observations.

The monoid $\mathcal{F}$ induces a morphism *flow* which maps discrete updates to 1_a for $a \in \mathcal{W}$ and elements of Σ to ϵ respectively (with ϵ the zero element of Σ^*). The result is a natural concatenation of flows in distinct time intervals.

This project focuses on using the algebraic framework in order to define notions of regularity and also arrive at exact definitions for the Resource Systems lying on the frontier of decidability for (time bound) reachability which is expected to be strictly related to regularity.

Bibliography

[1] Eugene Asarin, Paul Caspi, and Oded Maler. "Timed Regular Expressions". In: *Journal of the ACM* 49 (Dec. 2001). DOI: 10.1145/506147.506151.

[2] Giovanni Bacci et al. "Optimal and Robust Controller Synthesis". In: *Formal Methods*. Ed. by Klaus Havelund et al. Cham: Springer International Publishing, 2018, pp. 203–221. ISBN: 978-3-319-95582-7.

[3] Matthias Boehm et al. "Data Management in the MIRABEL Smart Grid System". English. In: *Proceedings of the 2012 Joint EDBT/ICDT Workshops, Berlin, Germany, March 30, 2012.* null ; Conference date: 30-03-2012 Through 30-03-2012. United States: Association for Computing Machinery, 2012, pp. 95–102. ISBN: 978-1-4503-1143-4. DOI: 10.1145/2320765.2320797.

[4] Stijn Cole et al. "Energy storage on production and transmission level: a SWOT analysis". In: Jan. 2005.

[5] Alexandre David et al. "Uppaal Stratego". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Christel Baier and Cesare Tinelli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 206–211. ISBN: 978-3-662-46681-0.

[6] Jonas Hansen, Kim Guldstrand Larsen, and Pieter J.L. Cuijpers. "Balancing Flexible Production and Consumption of Energy using Resource Timed Automata". In: *2022 11th Mediterranean Conference on Embedded Computing (MECO)*. 2022, pp. 1–6. DOI: 10.1109/MECO55406.2022.9797191.

[7] Thomas A. Henzinger. "The Theory of Hybrid Automata". In: *Verification of Digital and Hybrid Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 265–292. ISBN: 978-3-642-59615-5. DOI: 10.1007/978-3-642-59615-5_13. URL: https://doi.org/10.1007/978-3-642-59615-5_13.

[8] Nick Jenkins et al. *Smart Grid*. Wiley., 2012.

[9] Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. "Safe and Optimal Adaptive Cruise Control". In: *Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015, Proceedings*. Cham: Springer International Publishing, 2015, pp. 260–277. ISBN: 978-3-319-23506-6. DOI: 10.1007/978-3-319-23506-6_17. URL: https://doi.org/10.1007/978-3-319-23506-6_17.

[10] Torben Bach Pedersen, Laurynas Šikšnys, and Bijay Neupane. "Modeling and Managing Energy Flexibility Using FlexOffers". In: *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. 2018, pp. 1–7. DOI: 10.1109/SmartGridComm.2018.8587605.

[11] *Project Mirabel*. https://mirabel.daisy.aau.dk/. 2013.

[12] Klaus Trangbaek et al. "Exact power constraints in smart grid control". In: *Proceedings of the IEEE Conference on Decision and Control* (Dec. 2011), pp. 6907–6912. DOI: 10.1109/CDC.2011.6160539.

[13] Emmanouil Valsomatzis, Torben Pedersen, and Alberto Abelló. "Day-ahead Trading of Aggregated Energy Flexibility". In: June 2018, pp. 134–138. DOI: 10.1145/3208903.3208936.

[14] Matteo Vasirani and S. Ossowski. "Smart consumer load balancing: State of the art and an empirical evaluation in the Spanish electricity market". In: *Artificial Intelligence Review* 39 (Jan. 2013). DOI: 10.1007/s10462-012-9391-6.

[15] Laurynas Šikšnys and Torben Bach Pedersen. "Dependency-Based FlexOffers: Scalable Management of Flexible Loads with Dependencies". In: *Proceedings of the Seventh International Conference on Future Energy Systems*. e-Energy '16. Waterloo, Ontario, Canada: Association for Computing Machinery, 2016. ISBN: 9781450343930. DOI: 10.1145/2934328.2934339. URL: https://doi.org/10.1145/2934328.2934339.

[16] Laurynas Šikšnys et al. "Aggregating and Disaggregating Flexibility Objects". In: *IEEE Transactions on Knowledge and Data Engineering* 27.11 (2015), pp. 2893–2906. DOI: 10.1109/TKDE.2015.2445755.