Modelling Real-time Systems with Bigraphs

Maram Albalwe
University of Glasgow
Glasgow, UK
University of Tabuk
Tabuk, Saudi Arabia
m.albalwe.1@research.gla.ac.uk

Blair Archibald Michele Sevegnani
University of Glasgow
Glasgow, UK
{blair.archibald, michele.sevegnani}@glasgow.ac.uk

Bigraphical Reactive Systems (BRSs) are a graph-rewriting formalism describing systems evolving in two dimensions: spatially, *e.g.* a person in a room, and non-spatially, *e.g.* mobile phones communicating regardless of location. Despite use in domains including communication protocols, agent programming, biology, and security, there is no support for real-time systems. We extend BRSs to support real-time systems with a modelling approach that uses multiple perspectives to represent digital clocks. We use Action BRSs, a recent extension of BRSs, where the resulting transition system is a Markov Decision Process (MDP). This allows a natural representation of the choices in each system state: to either allow time to pass or perform a specific action. We implement our proposed approach using the BigraphER toolkit, and demonstrate the effectiveness through multiple examples including modelling cloud system requests.

1 Introduction

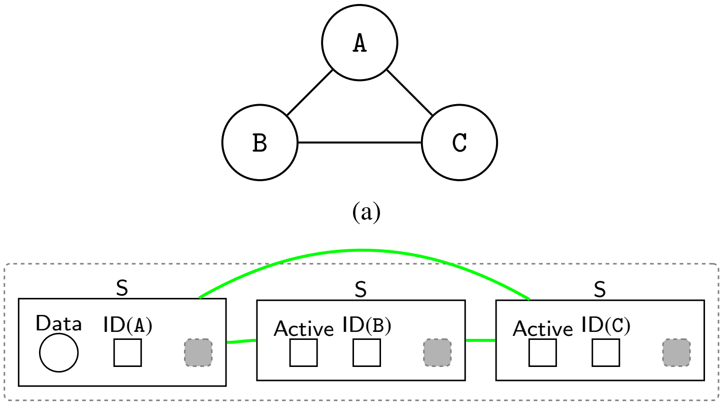
Bigraphs are a computational model where systems are described based on two types of relationships: spatially using *nesting* (and parallel adjacency) and non-local linking through hyperlinks regardless of locations. Like standard graphs, bigraphs have an equivalent diagrammatic and algebraic representation. Bigraphical Reactive Systems (BRSs) equip bigraphs with a set of reaction rules that specify how a system evolves over time, *i.e.* reaction rules substitute sub-bigraphs with other bigraphs.

The standard theory of bigraphs has been extended to model a wider range of systems *e.g.* stochastic bigraphs [11] assign rates to reaction rules, bigraphs with sharing [21] allow intersecting locations, directed bigraphs [8] associate directions to links, conditional bigraphs [2] add conditions to rules, and probabilistic and action bigraphs [3] support probabilistic and non-deterministic behaviour. Using these extensions, bigraphs have been successfully used to model a variety systems including mixed-reality games [6], cloud systems [19], self-adaptive fog systems [20], sensor network infrastructure [23], and rational agents [4].

An extension that has not been explored is real-time bigraphs that can model systems exhibiting non-deterministic behaviour that is controlled by time constraints. While the non-deterministic aspects of real-time systems could be modelled by action bigraphical reactive systems (ABRSs) in which the underlying Markov Decision Process (MDP) semantics allows for a transition choice at each state, there is currently no notion of *clock constraints* in the theory, *e.g.* specifying that after 3 time units have passed, a given action *must* occur.

We propose a modelling strategy to encode timed systems within ABRSs through the well-known MDP-based digital clock approximation. We introduce a clock, as a new bigraph entity, for each timed entity in the system and we place these in a separate region so all the clocks can be manipulated without polluting the actual system model. This allows one reaction rule to advance all clocks at once, and this reduces the state space overhead of the approach. We mirror the real-time passage by introducing a

J. Endrullis, D. Grzelak, T. Heindel and J. Kosiol (Eds):
GCM 2023 and GCM 2024
EPTCS 417, 2025, pp. 96–116, doi:10.4204/EPTCS.417.6



(a)

(b)

Figure 1: (a) Example network topology with three sensors; (b) corresponding bigraph with data to be transmitted by sensor A and the status of each sensor (*e.g.* Active).

global clock that controls the system execution time, *i.e.* system terminates when we reach the time limit of this clock, and we explicitly model the non-deterministic behaviour, *i.e.* at each state there is a choice between taking an action or allowing time to pass when the constraints are not yet satisfied.

We make the following research contributions:

- We propose a modelling strategy to express clocks within action bigraphs.
- We define a set of reaction rules to model clock constraints in real-time systems through a digital clocks approximation.
- We illustrate how to use our approach in practice by partially modelling the timed aspects of a cloud computing scenario.
- We implement this approach in BigraphER [22] to show this is not just a theoretical contribution, but a practical one.

Outline. We give an informal description of bigraphs and BRSs in Section 2. Section 3 provides a description of non-deterministic models and shows how to formalise digital clocks within ABRSs. We illustrate our approach by providing a BRS implementation for a simple scenario modelled as a (probabilistic) timed automaton and a model of cloud system requests in Section 4. Section 5 gives a brief literature review, and we conclude in Section 6.

2 Background

2.1 Bigraphs

Introduced by Milner [16], Bigraphs provide a powerful diagrammatic representation for modelling systems that evolve in both spatial and non-spatial dimensions. Bigraphs represent the structure of a system through two relations over its entities: a *place graph* that encodes their spatial arrangement, *e.g.* a person in a room, and a *link graph* that specifies, through hyper-edges, non-spatial relations, *e.g.* communication capabilities between network devices. We give an informal description of bigraphs using the example in

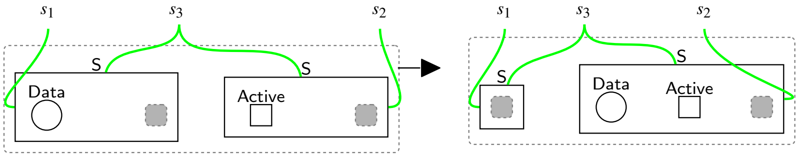


Figure 2: Reaction rule `send_Data` applies when the receiver is active.

Figure 1. A comprehensive formal description can be found elsewhere [16]. Bigraphs have both a diagrammatic representation and *equivalent* algebraic notation. We use the diagrammatic notion throughout this paper.

Figure 1a shows a simple network topology consisting of three sensors connected to each other by communication links. Data may be exchanged between two sensors when the receiver is active. A corresponding bigraph representation is in Figure 1b. Sensors are represented as *entities* of type S that have *nested* identifiers (also just a different type of entity) ID(`A`), ID(`B`), and ID(`C`). The sender contains an entity Data that can be sent to other sensors when they are in the Active mode. We sometimes draw different entity types by using different shapes or colours *e.g.* we have used squares for identifiers and a circles for Data. Entities can be *atomic*, *e.g.* Data, meaning they have no children, or contain any number of other entities. We allow entities to be parameterised to represent families of entities, *i.e.* ID(x) specifies a new entity for every possible value of x (which may be string, integer, or float typed). Sites—filled dashed rectangles—represent unspecified bigraphs: an arbitrary bigraph, including the empty bigraph, might exist there. Bigraphs may consist of more than one region—clear dashed rectangles—which represent adjacent parts of the system. For the link graph, entities have a fixed *arity* that represents the number of *links* they must have (the green edges). Links are hyperlinks that may connect 1-to-n. Open links—a link that has a name— indicates a link that may connect elsewhere, *i.e.* to currently unspecified entities. Links may also be closed, a 1-to-0 hyperedge, which is shown by an orthogonal line at the end of the link. Open names, sites, and regions allow model composition: regions of one bigraph can be placed in the sites of another and like-names joined. This forms the basis of the rewriting theory.

2.2 Bigraphical Reactive Systems (BRSs)

A bigraph represents a system at a single point in time. To model dynamic behaviour, Bigraphical Reactive Systems (BRSs) equip bigraphs with a set of *reaction rules* specifying how the system may evolve. Reaction rules take the form $L \longrightarrow R$ meaning that when the rule is applicable to a state S (state S *matches* bigraph L) the system evolves by replacing an occurrence of bigraph L in S with bigraph R. BRSs form a transition system that has an initial state (*bigraph*), and the transition from one state (*bigraph*) to another is defined by generating all possible rewrites (*reactions*). For example, the rule in Figure 2 is applicable where there is a sensor that has Data to send to another Active sensor in its range.

We control the execution of a set of reaction rules via *priority classes*. That is $\{\mathtt{r}_1, \mathtt{r}_2\} < \{\mathtt{r}_3\}$ means rules $\mathtt{r}_1$ and $\mathtt{r}_2$ can be applied only if it is not possible to apply $\mathtt{r}_3$. Similarly to entities, rules can be parameterised to allow multiple rule applications over a predefined set of values.

3 Modelling Time with Action Bigraphs

The big idea is that by utilising Action Bigraphs [3] we can encode timed systems. This is based on a well-known approximation of (probabilistic) timed automata to Markov Decision Processes [10]. Intuitively, we extend the usual action semantics to allow two forms of action: *discrete actions* that encode system events (which may only be enabled at some time), and *time* actions that progress time.

Action Bigraphical Reactive Systems (ABRSs) allow a choice of probability distributions at each rewriting step by assigning *weights* to the reaction rules, and by giving action labels to specific sets of reaction rules. An action is enabled/possible whenever there is a reaction rule from the set which is enabled. The resulting transition system is a Markov Decision Processes (MDP) [17] which can be verified using off-the-shelf model checkers such as PRISM [12]. An MDP is a tuple $(S, s_0, A, Step)$ where S is a set of states with a predefined initial state $s_0 \in S$. For each state, we can choose an (enabled) action $a \in A$ which gives an associated probability distribution over future states as specified by $Step$.

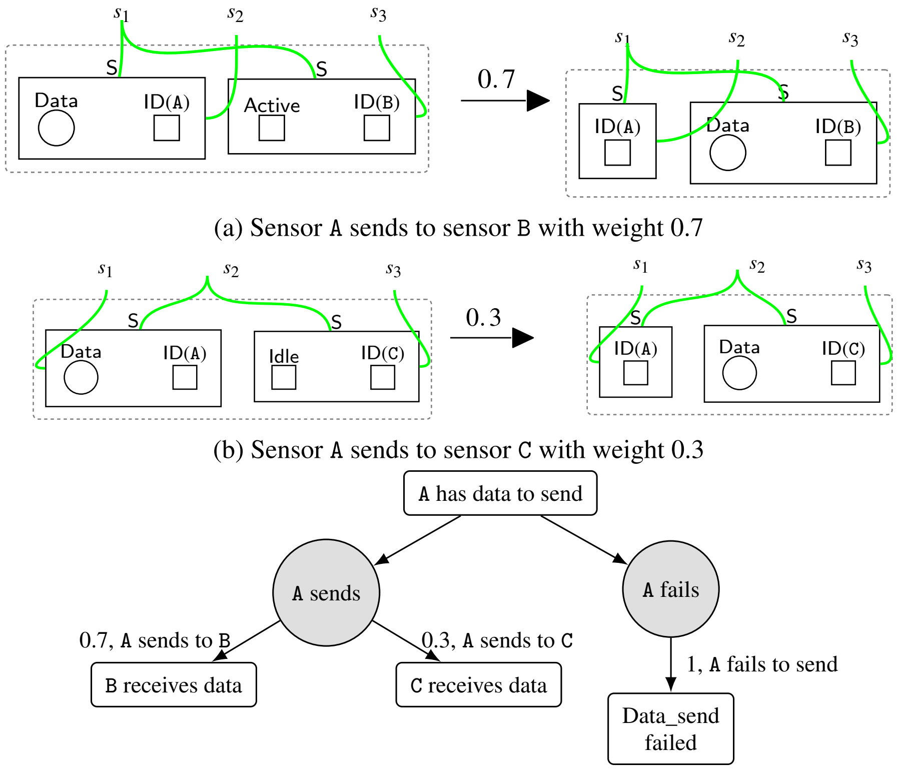
To show how ABRSs model non-deterministic behaviour of real-time systems where underlying transition system is an MDP, we use the bigraph example shown in Figure 1b. That is for sensor A, data can either be sent successfully to other sensors or fail to send. We also can permit sensor A to send Data with a bias by replacing the send_Data rule (Figure 2) with the two weighted (as shown in Figure 3). For example, given $weight = 0.7$ to B and $weight = 0.3$ to C, then sending to B is more than twice as likely. We use *bigraphs* throughout this paper to mean *Action Bigraphical Reactive Systems (ABRSs)*. Using BigraphER [22], an open-source toolkit for working with bigraphs (and the supporting ABRSs), we can export the corresponding MDPs transition system to be formally checked. Importantly, action bigraphs still respect any rule priorities. This means an action will fire with any high priority rule before firing with those of lower priorities. We use this feature in our encoding of clock invariants.

While MDPs support uncertainty modelling for dynamic environments, Probabilistic Timed Automata (PTAs) [1] extend this by incorporating timing constraints which offer a more comprehensive framework for modelling and reasoning about real-time systems. A PTA is defined as a tuple in the form (L, l_0, E, A, C, I) where L is a set of locations (*i.e.* states) with $l_0 \in L$ a start location. E is a set of edges that represents the possible transitions between locations while A is a set of (discrete) actions that may occur in the system. C is a finite set of clocks and I is a set of clock invariants associated to each location. While these invariants could be flexible, practically they are typically used to force a transition from a location, *e.g.* an invariant $x \leq 2$ forces a location move at time 2 if we have not already left the location (alongside transition conditions these usually encode a "move within a maximum of 2 time units" semantics). Transitions are associated to actions and probability distributions, with the addition of *clock constraints* (*e.g.* $x > 3$) and *clock resets* (*e.g.* $x := 0$).

In the *digital clocks approximation* we give the semantics of PTAs as a Markov Decision Process where states (including initial) are all the locations where clock invariants are met, and we form a single action set consisting of both actions manipulating time and user-specified discrete actions as follows:

- a *discrete* action is enabled if all clock constraints (given in E) associated with the transition are satisfied;
- a *time* action is available while invariants associated to $l \in L$ are satisfied as time elapses. That is, we cannot progress time if something must happen beforehand.

Since we use action bigraphs, we can associate discrete transitions with a probability distribution. If required we can, for example, draw with uniform probability to recover semantics for a non-probabilistic timed automata.



(a) Sensor A sends to sensor B with weight 0.7

(b) Sensor A sends to sensor C with weight 0.3

(c) MDP generated by the ABRS.

Figure 3: Example Action Bigraphical Reactive System: probabilistic reaction rules using weights.

3.1 Digital Clocks in Bigraphs

Standard BRSs evolve to a new state whenever there is a match of a reaction rule[1]. In this sense they are non-deterministic but not explicitly so, *i.e.* we cannot label particular actions without ABRSs. Models of real-time systems need to both find enabled matches, and meet any requirements introduced by clock constraints. We propose a modelling technique that encodes digital clocks as *entities* to model real-time systems. Although PTAs can work with real-valued clocks, for the digital clocks approximation we fix clock values to non-negative integers and bound the total runtime of the system (to ensure a finite action set which means the models can be analysed through existing MDP reachability techniques). As for the standard digital clocks approximation, our approach does not support strict inequalities and comparisons between clocks. As we have action bigraphs, these clocks can live within the bigraph model itself, and a separate type of semantics is not needed (unlike for probabilistic bigraphs for example). We show our approach by an example, and the main idea is in Figure 4. We put all clocks into their own parallel region which we call the *clocks perspective*. A global clock is represented by an entity family $\mathsf{GC}(n)$—*i.e.* there is one entity for each $0 \leq n \leq t_{\text{max}}$ for some max time t_{max}—and is used to model *wall clock time*. Wall-time cannot be reset. Local clocks are entity families $\mathsf{LC}(n)$ where $0 \leq n \leq t_{\text{max}}$ for each timed entity. For clarity of modelling, we place these in LocalClocks, although this is not strictly required. We identify a set of *timed* entities and expand their arity by 1 to link them to a specific local clock. Using this, we can *identify* a clock through the linked entity, *i.e.* clocks do not need specific identifiers of their own. Not

[1] Subject to any requirements about rule priorities and conditions.

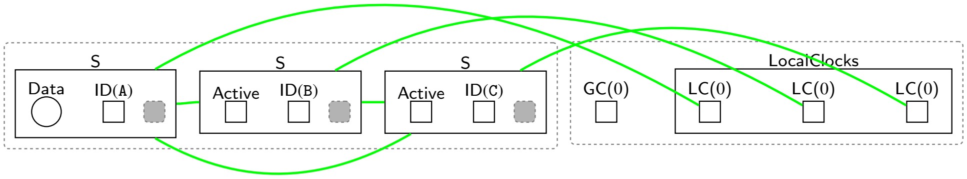


Figure 4: Example from Figure 1b extended with clocks perspective.

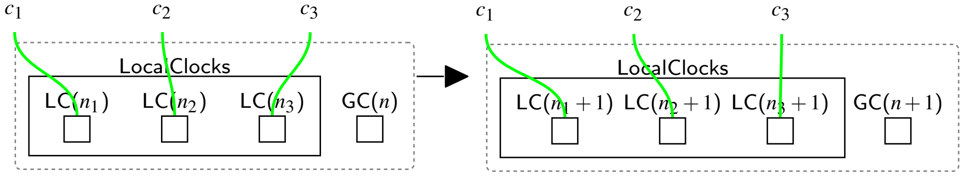


Figure 5: Reaction rule $\texttt{clock\_advance}(n_1,n_2,n_3,n)$ for a system with three timed entities.

all entities in the system will own a clock, *e.g.* Data is not timed. We assume all clocks are initialised to 0 in the initial state.

The approach to place clocks within their own region is a design choice, and because of the flexibility of bigraphs, other choices would be possible. For example, we could *nest* local clocks within particular timed entities instead of linking to them. We chose the extra region as it does not clutter any existing model, *i.e.* we can convert an existing model to a timed representation without significant changes (other than arity) in existing regions.

3.2 Reaction Rules for Digital Clocks

In our digital clocks approximation we have two main types of action: *discrete* actions which are user-defined and system specific, and *time* actions that deal with the passage of time. As actions in ABRSs are modelled as sets of reaction rules, the main challenge here is defining appropriate reactions for each type of action.

For timing actions, the main rule is $\texttt{clock\_advance}(n_1,n_2,\dots,n)$ shown in Figure 5 (for a system with three timed entities). This rule advances all local and global clocks simultaneously. All clocks advance at the same speed (one unit per application). This is a parameterised rule, which, like parameterised entities, defines a family of rules *i.e.* one for each possible value of the parameters. The *tick* action consists of the set of all possible $\texttt{clock\_advance}(n_1,n_2,\dots,n)$ rules for parameters drawn from $n,n_1,n_2,\dots \leq t_{\text{max}}$, for some max time t_{max}. Due to the parameters only one instance of clock_advance will ever be enabled meaning this action always advances time with probability one.

For *discrete* actions we extend existing system rules whenever a time constraint must be met. We take rules that have a standard (untimed) definition of a bigraphs rule ($L \longrightarrow\!\!\blacktriangleright R$) and, like with the clocks perspective, utilise parallel regions to link timed entities with their clocks as follows:

$$/c\,(\widehat{L_c} \parallel \mathsf{LC}(n)_c) \longrightarrow\!\!\blacktriangleright /c\,(\widehat{R_c} \parallel \mathsf{LC}(n')_c)$$

where $n' \in \{n, 0\}$, notation $\widehat{\ }$ indicates the transformation over bigraphs described in Section 3.1 in which the entity type of timed entities has its arity increased by one, *i.e.* each entity has one more name. The

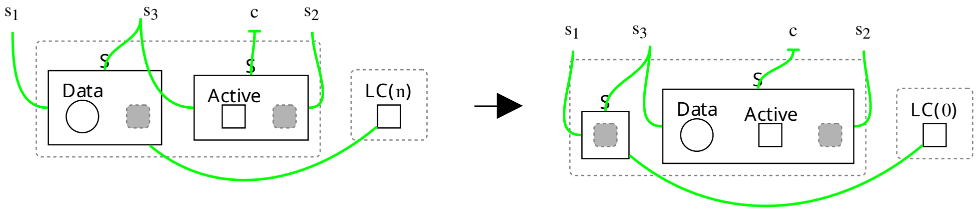


Figure 6: Reaction rule(s) `sending_data`(n). Time constraints are encoded by setting $2 \leq n \leq t_{\max}$.

subscript c indicates the name used to link a specific timed entity to a new clock placed in a *different* regions as specified by the $||$ operator. Other entities are forbidden from linking over c as the link is *closed* (with $/c$). Importantly, this change also makes existing rules into parameterised rules (over parameter n). This is used later to encode timing constraints, *e.g.* we chose a set of parameters that define at what (local) *time* a particular rule can be applied. For rules that are already parameterised, we can simply add an additional parameter for the clock.

As an example we extend our rule in Figure 2, that allows sensors to send data, to only fire when the receiver is active, *and* at least two time units have passed *e.g.* $\mathsf{LC}(n) \geq 2$. As we are modelling an inequality we cannot do this with a single reaction rule and instead we provide a new parameterised rule `sending_data`(n), shown in Figure 6, that explicitly links the sending sensor to its local clock. The rule is only valid for $n \in \{2,3,\ldots,t_{\max}\}$ so this is the family of rules we generate. In this case, we include a clock reset on the right-hand-side of the rule, and clocks may only be reset during the application of a rule.

Note that the `clock_advance` rule has the same priority as the corresponding rules whose applications are triggered by clock constraints. This allows time to pass until the clock valuation satisfies the first invariant. In a such state, the non-deterministic choices are applicable *i.e.* the system can take a discrete action or permit the time to pass if it is not the last valid clock valuation in which an action must occur.

We must also encode any location-based clock invariants, *e.g.* locations that are only valid for $x \leq 2$. In practice this is used to force a transition to occur rather than allowing an indefinite wait within a particular location, and is almost always *true* (allowing indefinite waits) or a $\leq$ constraint. To encode these invaraints we make use of priority classes in the ABRS. For a state invariant, *e.g.* $t \leq x \leq n$, we add any outgoing transition rules $\mathtt{r}(n)$ such that $\{\mathtt{clock\_advance}(\ldots),\mathtt{r}(t),\mathtt{r}(t+1),\ldots,\mathtt{r}(\mathtt{n}-1)\}\} < \{\mathtt{r}(n)\}$. This means $\mathtt{r}(n)$ applies *before* any clock updates and *forces* a state transition before the invariant is broken.

4 Examples and Implementation

We implement our approach in BigraphER which also generates the MDPs (that are the semantics for our digital clocks representation). The exported MDPs can be formally checked using standard (probabilistic) model checkers, *e.g.* PRISM, allowing us to benefit from existing model checking algorithms.

Modelling a timed system with our approach follows these general steps:

- Define sets of clock valuations/invariants according to the system requirements.
- Define a new clock perspective and add clock entities equal to the number of the timed entities required, and one global clock. Increase the arity of timed entities by 1, and link to the local clock. All clocks are initialised to 0.

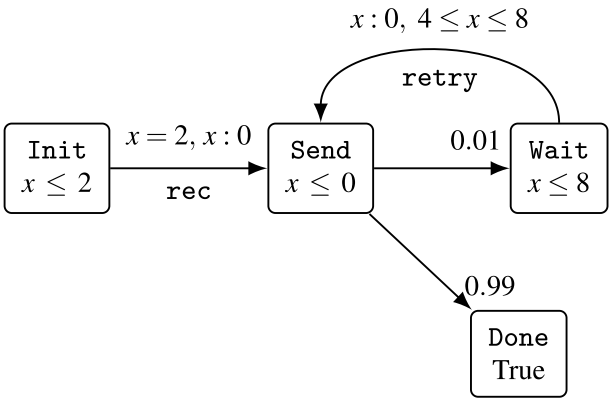


Figure 7: A probabilistic timed automata example model for a simple sending data process from [13].

- For a given $t_{\max}$ and number of timed entities, generate the set of rules in the form

$$\texttt{clock\_advance}(\{0,\ldots,t_{\max}\},\{0,\ldots,t_{\max}\},\ldots)$$

- Extend system rules by adding in the clock perspective when required to restrict their application based on the time constraints. Parameters for these rules must meet the clock invariants. Any state-based invariants are encoded using priorities.

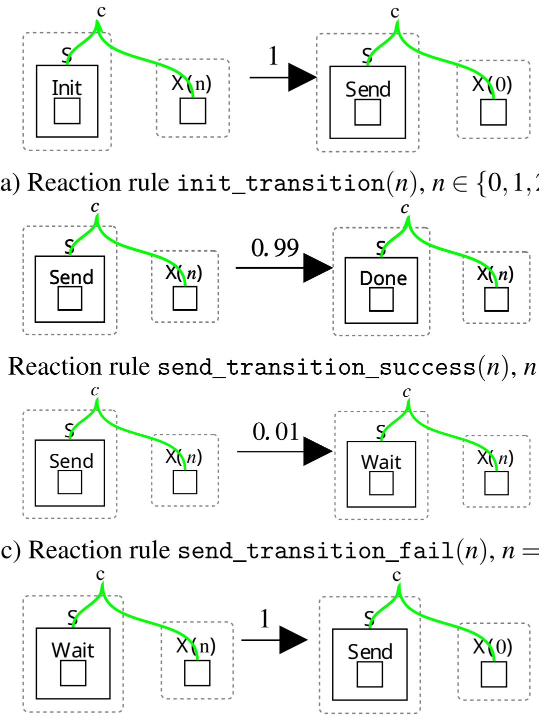
We illustrate our approach by encoding two different examples: a probabilistic timed automata example, and requests allocation in a cloud system [14]. The BigraphER models for both examples are in the appendix.

4.1 PTA example

We apply our approach to the probabilistic timed automata example shown in Figure 7 and recreated from [13]. This simple communication system attempts to send data based on the constraints over clock x. Here we do not model the actual sends, only the state machine the system goes though. This is somewhat unnatural for bigraphs, where we are more concerned with global system updates (possibly of multiple agents) than encoding a particular state machine for an entity, but is used to illustrate that we can recover existing PTA semantics.

The system starts in the initial state `Init`. When the time has elapsed exactly 2 time units (as forced by the transition constraint *and* the clock invariant on the state) the system moves to the `Send` state. The clock invariant $x \leq 0$ forces data to be sent immediately. With probability 0.99 the system reaches its final state, `Done`, and with 0.01 probability it fails and moves to a `Wait` state. The system waits at least 4 time units and at most 8 time units (forced by the invaraints) before moving back to the `Send` state to retry. The clock is reset with each transition.

We start modelling this PTA example by defining the required time constraints. Here we use $\mathsf{X}(n)$ instead of $\mathsf{LC}(n)$ to be closer to the original example. For each state we define the valid clock valuations as follows. `init_transition` rule (Figure 8a) applies over $n \in \{0,1,2\}$ for `Init` state, and the rules that are associated to `Send` state (Figure 8b and Figure 8c) fire immediately upon receiving data *i.e.* at $n = 0$. While `wait_transition` rule (Figure 8d) is applicable over $n \in \{4,5,6,7,8\}$. For all parameters except 2 for `Init` and 8 for `Wait` states, the rules application should be in the same priority class as `clock_advance`$(\ldots)$ rule allowing the non-deterministic behaviour: time passes or an data



(a) Reaction rule `init_transition`(n), $n \in \{0,1,2\}$

(b) Reaction rule `send_transition_success`(n), $n = 0$

(c) Reaction rule `send_transition_fail`(n), $n = 0$

(d) Reaction rule `wait_transition`(n), $n \in \{4,5,6,7,8\}$

Figure 8: Reaction rules for the PTA example.

sends. However, as the invariants force leaving if more than $t_{\max}$ units elapse, `init_transition`(2) `send_transition`(0), and `wait_transition`(8) are in a higher priority class. We encode a single rule for each system transition, and show how the probabilities are encoded using rule weights[2], *e.g.* to allow `Send` state to move to either `Wait` or `Done`.

In Figure 9, we show the clocks bigraph model conforms to the probabilistic timed automata example by giving the transition system. For space, we only provide the first few transitions. Here the system moves from the initial state `Init` to the `Send` state. When the system is in the `Init` state and the clock constraints are satisfied, there are two possibilities: the time passes or an action occurs. Once the clock becomes X(2), the system **has to** move to `Send`.

4.2 Cloud System

We remodel the example presented in [14]. The authors also add time constructs to BRSs, but their work differs as it expresses clocks as nested entities and needs multiple reaction rules to advance clocks (which can cause unnecessary state-space explosion). They do not have a method to track approach wall-time and, as they do not use action BRSs, cannot model the (explicit action) non-deterministic behaviour of many real-time systems.

A cloud system is a collection of resources that include hardware, software, networks, etc. that is used to store, manage, and process data. That is end users (EU) send their requests (R) over the internet

[2] In this case, as there is only one agent moving state, the weights are equivalent to their probabilities since there will never be additional rule application matches to account for.

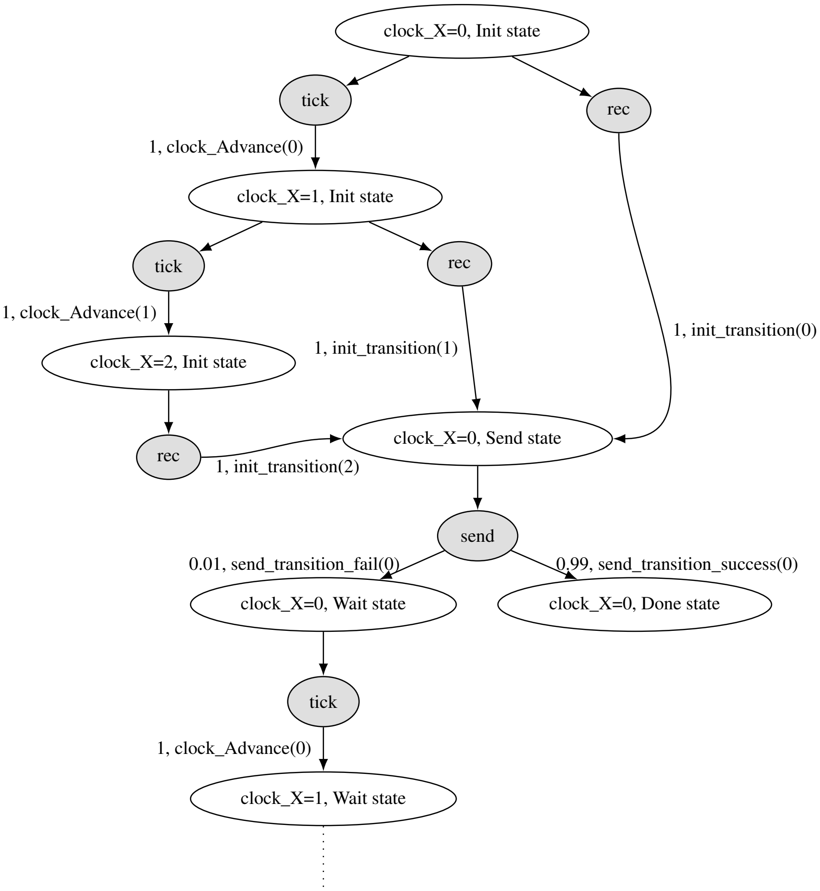


Figure 9: Resulting MDP (partial) for the probabilistic timed automata shown in Figure 7.

to virtual machines (VM) hosted on physical servers ($\mathsf{S}(i)$ with $i > 0$) within a data centre (DC) for processing. Standard bigraphs can model and analysis the structure and dynamic behaviour of cloud systems but not the time constraints that cloud applications usually have *e.g.* the duration of tasks. We consider a system that has two end users connected to each other via a link *e.g.* e_1; and two servers that are also connected via a link *e.g.* e_2. A data centre and an end user relate to each other via a link *e.g.* y_1. There are four requests each of them connects to its end user through a link *e.g.* x_1. The requests need to be processed using two servers. We model the same example as follows. We give each request a parameterised identifier ($\mathsf{ID}(r)$, where $r \in 1,2,3,4$). We also associate each request with different controls to reflect its current status. Initially all requests are in a (Wait) status; requests that are under processing have status Processing while status Result indicates that a request has been processed and returned to the end user. We define a clock for each request but, unlike in [14], we add a global clock that shows the wall-time. We also use a reaction rule that simultaneously advances all the clocks. We specify one reaction rule `sendingRequest` (Figure 10a) to send a request from an end user to a virtual machine for processing. Given that the servers have varying resources and thus require different amounts of time to process a request, two rules are applied to return the requests after processing, *i.e.* once a predefined time has elapsed. Rule `processRequest_S1` is for server $\mathsf{S}(1)$ that takes 2 time units and rule `processRequest_S2` is for server $\mathsf{S}(2)$ that needs 3 time units to process a request. We define the same time constraint assumptions stated in [14] as integer sets. These sets are to be utilised with the timed reaction rules. That is, for each request there are four integer sets as follows. The *first one* is the valid clock valuations for a request. The *second* contains the time at which a request can be sent. The *last one* is two different sets that determine the time that a request should last according to the processing time for each server. We provide the four integer sets as follows:

- $N = \{n \mid 0 \leq n \leq t_{\max}\}$ for each request,
- $S = \{1,3,4,5\}$ for requests 1, 2, 3, and 4 respectively,
- $P_1 = \{s+2 \mid s \in S\}$ for requests processed on server $\mathsf{S}(1)$,
- $P_2 = \{s+3 \mid s \in S\}$ for requests processed on server $\mathsf{S}(2)$.

While in [14] virtual machine migrations are considered, here we allocate the process to a machine that is in Idle status indicating it is free and ready to receive a request. We model sending requests through reaction rule `sendingRequest` (Figure 10a). This rule allocates the request to an Idle server when a request's time constraint is satisfied. We then encode two rules that return requests back to the sender. Note that depending on the allocated server, only one of the return rules applies: rule `returnRequest_S1` takes the result back to EU for requests that are treated on server $\mathsf{S}(1)$ once 2 time units elapsed; while rule `returnRequest_S2` takes it back to EU after 3 time units passes for requests that are treated on server $\mathsf{S}(2)$. Figure 10b shows a return rule that applies when the time constraint (either p_1 or p_2) is met for $\mathsf{S}(i)$. We will show in the following section how our approach can be used to verify the interesting properties. We make the assumption that a request is always forced to send no more than 1 time unit *after* the deadline. This means processing time for delayed tasks can be faster than expected, *e.g.* $p_1 = s+2-1$. This could be accounted for using additional entities, *e.g.* Delayed, or out of time requests could be dropped. Bigraphs give the flexibility to experiment with these approaches with additional rules.

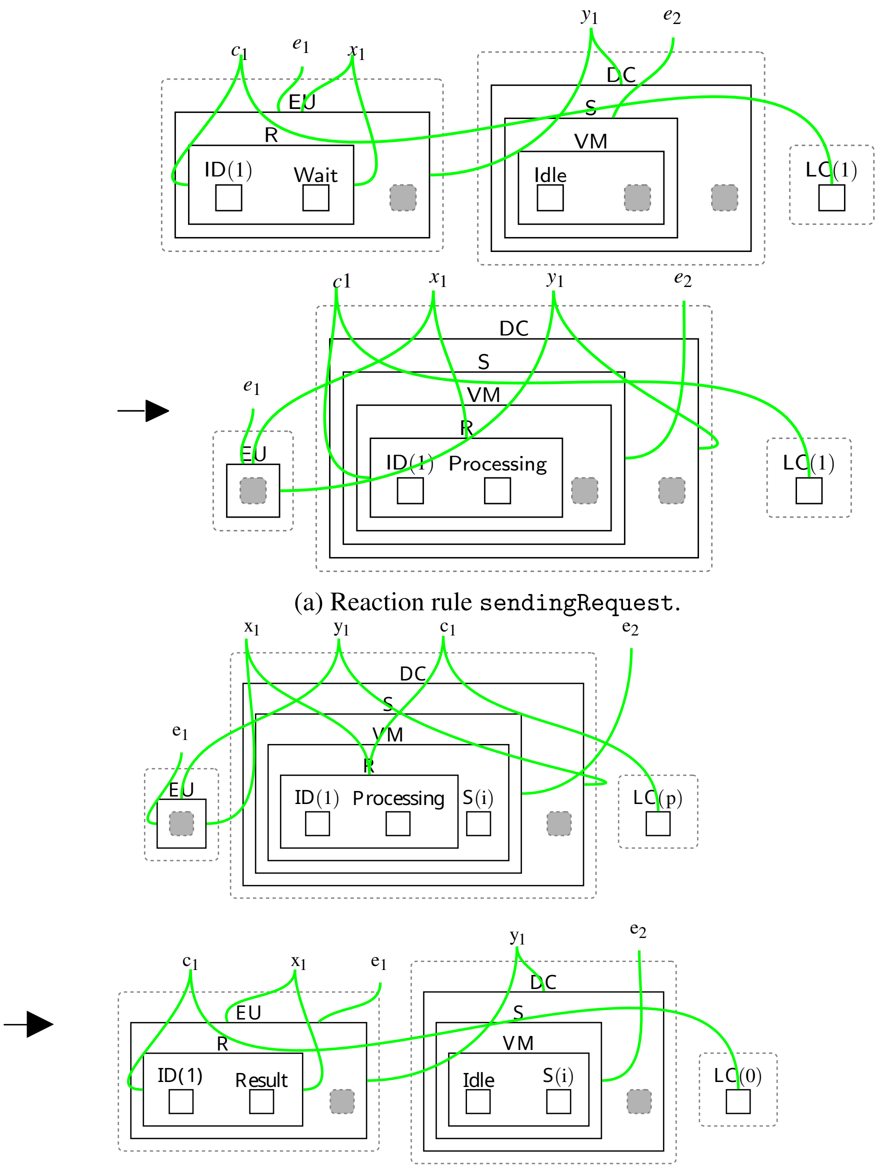
4.3 Model Analysis

To check the feasibility of our proposed modelling technique, we export the bigraph models into MDPs automatically using BigraphER. We can then verify them against required properties using PRISM by expressing properties in the Probabilistic Temporal Logic (PCTL) [9]. To help writing properties, we utilise *bigraph patterns* that allow states to be labelled based on matches [6], *e.g.* label any state that has a Data entity. Since clocks are just part of the model, we can also label clock values, *e.g.* $clock\_LC_0$ for $\mathsf{LC}(0)$. Interestingly, the clocks used in the properties are those in the model, *i.e.* they are just bigraph entities, rather than clock variables you would generate if you, for example, expressed the PTA directly in PRISM. This gives flexibility, *i.e.* we can write predicates over many different types of clock matches.

We make use of the following PCTL quantifiers to specify our properties: **A** (for all) and **E** (there exists), and path formulae, **F** (eventually), **X** (next), **U** (until) and $\wedge$ (and). We also use probability quantifier **P** to check the probability of reaching a specific state. For example, $\mathbf{P} \leq 0.5\,[\mathbf{F}\, stateA]$ means: (over all paths) is there at most a 0.5 probability that the system eventually reaches a state satisfying $stateA$, we label a state as $stateA$ using a bigraph pattern.

We conduct model checking on several interesting properties. In the PTA example (Section 4.1), we can perform probabilistic reachability properties such as "Is there at least a 0.99 percent probability that the system moves to `Done` state successfully", which holds for this system.

$$\mathbf{P} \geq 0.99\,[\,\mathbf{F}\,(\texttt{In\_Done\_state})\,] \tag{1}$$



(a) Reaction rule `sendingRequest`.

(b) Reaction rule `returnRequest`(p,i), $p = 3$ when $i = 1$ and $p = 4$ when $i = 2$.

Figure 10: Processing rules for the first request.

We can also utilise clock predicates to check properties relating to time *e.g.* "eventually there is at least a 99% chance that the system moves to the `Done` state and resets clock X", which is true for our system. Importantly, these are based on our clock entities, not time-bounded properties within PRISM which operate on their own time units.

$$\mathbf{P} \geq 0.99\,[\,\mathbf{F}\,(\,\texttt{In\_Done\_state} \wedge \texttt{clock\_X}_0)\,] \qquad (2)$$

We can check clock invariants are satisfied, *e.g.* that the system is not be in `Wait` state after 8 time units have elapsed:

$$\mathbf{A}\,[\,\neg\,(\mathbf{F}\,(\texttt{In\_Wait\_state})\wedge(\texttt{clock\_X}_9)\,)\,] \tag{3}$$

As expected this is true in all cases.

For the cloud system we can, for example, check the first request is always sent once *one* time unit has elapsed. As we use next (**X**), the request must be sent immediately before any other state transition.

$$\mathbf{A}\,[\,\mathbf{F}\;\texttt{clock}_1\wedge(\mathbf{X}\;\texttt{request}_1\texttt{\_sent\_to\_S})\,] \tag{4}$$

5 Related work

Many modelling formalisms have been extended to real-time systems by introducing clocks. Timed CCS [25] extends Milner's CCS by introducing the time notion to create concurrency models for real-time systems. Timed CCS introduces a variable to record time delays *e.g.* before a message arrives. Similar to our work, timed CCS considers the positive real number including zero as the time domain for convenience, but the model can deal with another numerical domain *e.g.* the natural numbers. Timed Petri nets [26] extends Petri nets to allow time triggered transitions. It uses tokens to allow transitions to start and then they are placed into the output when the firing process terminates. The authors use random variables with continuous or discrete probability distribution functions for the firing time. Timed π-Calculus [18] extends the π-Calculus with continuous time to describe and reason about concurrent Cyber-Physical Systems and real-time systems. An executable operational semantics of π-Calculus is developed in Logic Programming to model concurrency.

Graph Transformation Systems (GTS) has been extended into Probabilistic Timed Graph Transformation Systems (PTGTSs) which enable modelling and analysing structure dynamic and timed and probabilistic behaviour of embedded systems [15]. The clock is a typed node that is contained in a graph to identify the nodes that are used for time measurement only. In this work, PTGTSs is formally mapped into Probabilistic Timed Automata (PTA) where the Probabilistic Timed Structure (PTS) of the mapped PTGTSs is equal to the PTS of the resulting PTA, hence they both satisfy the same set of PTCTL properties. The obtained PTA can be checked using PRISM. However, the mapping process does not consider three aspects of the PTA. First, since the PTA does not consider valuations in the labelling function, constraints are ignored in the mapping process so the constraint should be true for any such atomic proposition. It also considers the PTGTS that does not show timelocks during its execution instead of finding and removing them when mapping PTGTS into PTA. Finally, the GTS state space that is constructed for PTGTS is up to isomorphism as preserved clock nodes are respected which may affect the state space finiteness of the resulting PTA. In our work, we bound the clock valuations which ensures finite transition systems. Additionally, in case that a system frequently resets its local clocks, the employment of the global clock guarantees the finiteness of the transition system. To prevent timelocks and so obtain a proper transition between reachable states, we assign the maximum states invariant valuation to the rules that are in a higher priority.

Bigraphs have been applied to model the location and connectivity of components of structure-aware mobile systems using BigrTimo that combines the rTiMO process algebra and bigraphs [24]. It uses real-time constraints to control actions by showing the waiting time for communication. The work in [14] explicitly encodes clocks as entities within bigraphs to model and reason about cloud applications, and shares some similarities with our approach as discussed in Section 4.2. It adds a set of clocks and a set of clocks constraints that are associated with nodes. It then utilises two different types of rules: (1) a set of reaction rules to advance all clocks, all clocks are advanced at the same speed; (2) instantaneous rules[3] that are executed only when there is a match and time constraints of one or many nodes are satisfied. These rules can also update the clock constraints and resets the clocks. However, when a new time constraint is satisfied, the time cannot elapse and another instantaneous rule may apply subsequently. In contrast, we encode timed aspects as action bigraphs resulting in an MDP transition system that explicitly models the non-deterministic behaviour of timed systems, that then can be formally checked by different model checking tools. The work uses Real-Time Maude language [7] and its TCTL model checker to implement and analyse the approach. State-space explosion is reduced here by employing a strategy that models all clocks as a separate region allowing us to use only one reaction rule to advance all clocks simultaneously. We imitate the wall-time by utilising the global clock entity.

6 Conclusion

Bigraphical Reactive Systems (BRSs) have been successfully used to model a wide range of systems but, until now, they had no explicit support for real-time systems. We overcome this limitation by introducing a modelling technique that uses the digital clocks approximation of (probabilistic) timed automata to encode timing aspects. This approach relies on Action Bigraphs, which have as a semantics Markov Decision Processes (MDPs), meaning we can express both probabilistic and timing behaviour within models. Using BigraphER, we encode the proposed strategy and verify the MDP corresponding to each model using PCTL model checking. Using two examples, we show our approach supports multiple clocks and the transition system obtained via rewriting faithfully encodes the digital-clock approximation of the behaviour of a real-time system.

Our approach suffers from the same limitations as the digital-clock approximation *i.e.* currently we do not support diagonal clocks and strict inequalities. We mitigate state space explosion by adopting the following two strategies. First, we bound clock valuations thus we obtain finite transition systems. Second, we allow the base time unit to be specified in each model, effectively allowing clocks to advance by multiple ticks in one step. Another limitation of our approach is that we assume clocks are always synchronised, *i.e.* they all progress at the same speed, which might not always be the case in scenarios like wireless sensor networks and IoT.

In future, we will develop syntactic support for clock constraints in the BigraphER language to generate real-time ABRS models like the ones considered in the paper. Sorting schemes [5] (type systems for bigraphs) will ensure correct separation of clocks and model entities. Syntax and sorts will ensure our extended set of reaction rules is correct by construction. We also aim to extend our approach to also support diagonal clocks and strict inequalities.

Acknowledgement

This work is supported by the UK EPSRC projects CHEDDAR (EP/X040518/1) and CHEDDAR Uplift (EP/Y037421/1), and an Amazon Research Award on Automated Reasoning.

[3] Silent rules that do not appear in an output transition system.

References

[1] Rajeev Alur & David L. Dill (1994): *A Theory of Timed Automata*. Theor. Comput. Sci. 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[2] Blair Archibald, Muffy Calder & Michele Sevegnani (2020): *Conditional Bigraphs*. In Fabio Gadducci & Timo Kehrer, editors: Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings, Lecture Notes in Computer Science 12150, Springer, pp. 3–19, doi:10.1007/978-3-030-51372-6_1.

[3] Blair Archibald, Muffy Calder & Michele Sevegnani (2022): *Probabilistic Bigraphs*. Form. Asp. Comput. 34(2), doi:10.1145/3545180.

[4] Blair Archibald, Muffy Calder, Michele Sevegnani & Mengwei Xu (2022): *Modelling and verifying BDI agents with bigraphs*. Sci. Comput. Program. 215, p. 102760, doi:10.1016/J.SCICO.2021.102760.

[5] Blair Archibald & Michele Sevegnani (2024): *A Bigraphs Paper of Sorts*. In Russ Harmer & Jens Kosiol, editors: Graph Transformation - 17th International Conference, ICGT 2024, Held as Part of STAF 2024, Enschede, The Netherlands, July 10-11, 2024, Proceedings, Lecture Notes in Computer Science 14774, Springer, pp. 21–38, doi:10.1007/978-3-031-64285-2_2.

[6] Steve Benford, Muffy Calder, Tom Rodden & Michele Sevegnani (2016): *On Lions, Impala, and Bigraphs: Modelling Interactions in Physical/Virtual Spaces*. ACM Trans. Comput. Hum. Interact. 23(2), pp. 9:1–9:56, doi:10.1145/2882784.

[7] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer & Carolyn L. Talcott, editors (2007): *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*. Lecture Notes in Computer Science 4350, Springer, doi:10.1007/978-3-540-71999-1.

[8] Davide Grohmann & Marino Miculan (2007): *Directed Bigraphs*. In Marcelo Fiore, editor: Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics, MFPS, LA, USA, April 11-14, 2007, Electronic Notes in Theoretical Computer Science 173, Elsevier, pp. 121–137, doi:10.1016/J.ENTCS.2007.02.031.

[9] Hans Hansson & Bengt Jonsson (1994): *A Logic for Reasoning about Time and Reliability*. Formal Aspects Comput. 6(5), pp. 512–535, doi:10.1007/BF01211866.

[10] Thomas A. Henzinger, Zohar Manna & Amir Pnueli (1992): *What Good Are Digital Clocks?* In Werner Kuich, editor: Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings, Lecture Notes in Computer Science 623, Springer, pp. 545–558, doi:10.1007/3-540-55719-9_103.

[11] Jean Krivine, Robin Milner & Angelo Troina (2008): *Stochastic Bigraphs*. In Andrej Bauer & Michael W. Mislove, editors: Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics, MFPS, PA, USA, May 22-25, 2008, Electronic Notes in Theoretical Computer Science 218, Elsevier, pp. 73–96, doi:10.1016/J.ENTCS.2008.10.006.

[12] Marta Z. Kwiatkowska, Gethin Norman & David Parker (2011): *PRISM 4.0: Verification of Probabilistic Real-Time Systems*. In Ganesh Gopalakrishnan & Shaz Qadeer, editors: Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings, Lecture Notes in Computer Science 6806, Springer, pp. 585–591, doi:10.1007/978-3-642-22110-1_47.

[13] Marta Z. Kwiatkowska, Gethin Norman, David Parker & Jeremy Sproston (2006): *Performance analysis of probabilistic timed automata using digital clocks*. Formal Methods Syst. Des. 29(1), pp. 33–78, doi:10.1007/S10703-006-0005-2.

[14] Fateh Latreche & Faiza Belala (2019): *Timed CTL checking of time critical cloud applications using timed bigraphs*. Int. J. Crit. Comput. Based Syst. 9(4), pp. 379–406, doi:10.1504/IJCCBS.2019.106818.

[15] Maria Maximova, Holger Giese & Christian Krause (2018): *Probabilistic timed graph transformation systems*. J. Log. Algebraic Methods Program. 101, pp. 110–131, doi:10.1016/J.JLAMP.2018.09.003.

[16] Robin Milner (2009): *The Space and Motion of Communicating Agents*. Cambridge University Press, doi:10.1017/CBO9780511626661.

[17] Martin L. Puterman (1994): *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics, Wiley, doi:10.1002/9780470316887.

[18] Neda Saeedloei & Gopal Gupta (2013): *Timed π-Calculus*. In Martín Abadi & Alberto Lluch-Lafuente, editors: *Trustworthy Global Computing - 8th International Symposium, TGC Argentina, August 30-31, 2013, Revised Selected Papers*, *Lecture Notes in Computer Science* 8358, Springer, pp. 119–135, doi:10.1007/978-3-319-05119-2_8.

[19] Hamza Sahli, Faiza Belala & Chafia Bouanaka (2015): *A BRS-Based Approach to Model and Verify Cloud Systems Elasticity*. In Keith G. Jeffery & Dimosthenis Kyriazis, editors: *1st International Conference on Cloud Forward: From Distributed to Complete Computing, October 6-8, 2015, Italy*, *Procedia Computer Science* 68, Elsevier, pp. 29–41, doi:10.1016/J.PROCS.2015.09.221.

[20] Hamza Sahli, Thomas Ledoux & Éric Rutten (2019): *Modeling Self-adaptive Fog Systems Using Bigraphs*. In Javier Cámara & Martin Steffen, editors: *Software Engineering and Formal Methods - SEFM 2019 Collocated Workshops: CoSim-CPS, ASYDE, CIFMA, and FOCLASA, Norway, September 16-20, Revised Selected Papers*, *Lecture Notes in Computer Science* 12226, Springer, pp. 252–268, doi:10.1007/978-3-030-57506-9_19.

[21] Michele Sevegnani & Muffy Calder (2015): *Bigraphs with sharing*. *Theor. Comput. Sci.* 577, pp. 43–73, doi:10.1016/J.TCS.2015.02.011.

[22] Michele Sevegnani & Muffy Calder (2016): *BigraphER: Rewriting and Analysis Engine for Bigraphs*. In Swarat Chaudhuri & Azadeh Farzan, editors: *Computer Aided Verification - 28th International Conference, CAV, ON, Canada, July 17-23, 2016, Proceedings, Part II*, *Lecture Notes in Computer Science* 9780, Springer, pp. 494–501, doi:10.1007/978-3-319-41540-6_27.

[23] Michele Sevegnani, Milan Kabác, Muffy Calder & Julie A. McCann (2018): *Modelling and Verification of Large-Scale Sensor Network Infrastructures*. In: *23rd International Conference on Engineering of Complex Computer Systems, ICECCS, Australia, December 12-14, 2018*, IEEE Computer Society, pp. 71–81, doi:10.1109/ICECCS2018.2018.00016.

[24] Wanling Xie, Huibiao Zhu & Qiwen Xu (2017): *BigrTiMo-A Process Algebra for Structure-Aware Mobile Systems*. In: *22nd International Conference on Engineering of Complex Computer Systems, ICECCS, Fukuoka, Japan, November 5-8, 2017*, IEEE Computer Society, pp. 50–59, doi:10.1109/ICECCS.2017.13.

[25] Wang Yi (1991): *CCS + Time = An Interleaving Model for Real Time Systems*. In Javier Leach Albert, Burkhard Monien & Mario Rodríguez-Artalejo, editors: *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Spain, July 8-12, 1991, Proceedings*, *Lecture Notes in Computer Science* 510, Springer, pp. 217–228, doi:10.1007/3-540-54233-7_136.

[26] Wlodek M Zuberek (1991): *Timed Petri nets definitions, properties, and applications*. *Microelectronics Reliability* 31(4), pp. 627–644, doi:10.1016/0026-2714(91)90007-T.

Appendix A BigraphER implementation of the examples in Section 4

```
#################  PTA Example #################

atomic fun ctrl X(n) = 1 ;
ctrl S = 1;
atomic ctrl Init = 0;
atomic ctrl Send = 0;
atomic ctrl Wait = 0;
atomic ctrl Done = 0;

#################  Reaction Rules #################

##Init
fun react init_transition(n) =
  S{c}.Init || X(n){c}
  -[1]->
  S{c}.Send || X(0){c};

## Send
fun react send_transition_success(n) =
  S{c}.Send || X(n){c}
  -[0.99]->
  S{c}.Done || X(n){c};

fun react send_transition_fail(n) =
  S{c}.Send || X(n){c}
  -[0.01]->
  S{c}.Wait || X(n){c};


# Wait
fun react wait_transition(n) =
  S{c}.Wait || X(n){c}
  -[1]->
  S{c}.Send || X(0){c};

#Done
react done_done =
  S{c}.Done -[1]-> S{c}.Done;

################# Clock Advance Rule #################
fun react clock_advance(n) =
  X(n){c}
  -[1]->
  X(n + 1){c};

################# Predicates #################
fun big clock_X(n) = X(n){c};
big in_Init_state = S{c}.Init;
big in_Send_state = S{c}.Send;
big in_Wait_state = S{c}.Wait;
big in_Done_state = S{c}.Done;

################# Initial State #################
big example_PTA = /c (S{c}.Init || X(0){c});

 begin abrs
  int n = {0,1,2,3,4,5,6,7,8};
  int maxInitT = {2};
  int init_Sending_Time = {0,1};
  int maxSendT = 0;
  int maxWaitT = 8;
  int wait_Sending_Time={4,5,6,7};

  init example_PTA;

  rules = [
    # Higher in the list => higher priority
    {done_done,
     init_transition(maxInitT),
     send_transition_fail(maxSendT),
     send_transition_success(maxSendT),
     wait_transition(maxWaitT)},

    {clock_advance(n),
     wait_transition(wait_Sending_Time),
     init_transition(init_Sending_Time)}
  ];

  actions = [
    send={send_transition_success,send_transition_fail},
    retry={wait_transition},
    rec={init_transition},
    deadlock={done_done},
    tick={clock_advance}
  ];

  preds = {
    in_Init_state,
    in_Send_state,
    in_Wait_state,
    in_Done_state,
    clock_X(n)
  };
end
```

```
#################  Cloud System Example #################

ctrl FrontEnd = 0;
ctrl EU = 3;
ctrl R = 2;
atomic ctrl Processing = 0;
atomic ctrl Result = 0;
atomic ctrl Wait = 0;
atomic ctrl Idle = 0 ;
atomic ctrl Stop = 0 ;
ctrl BackEnd = 0;
ctrl DC = 1;
ctrl S = 1;
atomic ctrl S1 = 0;
atomic ctrl S2 = 0;
ctrl VM = 0;
atomic fun ctrl ID(i) = 0;
ctrl LocalClock = 0;
atomic fun ctrl LC(requestClock) = 1;
atomic fun ctrl GC(globalClock) = 0;

################# Reaction Rules #################
fun react clock_advance(request1Clock, request2Clock, request3Clock, request4Clock, gc) =
  LocalClock.( LC(request1Clock){loclock1} | LC(request2Clock){loclock2} | LC(request3Clock){loclock3} | LC(request4Clock){loclock4} ) | GC(gc)
  -[1]->
  LocalClock.( LC(request1Clock +1 ){loclock1} | LC(request2Clock +1 ){loclock2} | LC(request3Clock +1 ){loclock3} | LC(request4Clock +1 ){loclock4} ) | GC(gc+1) if !Stop in ctx;

fun react sendingRequest(i, r1_Send_Time) =
  EU{x1,y1,e1}.(R{x1,c1}.(ID(i) | Wait ) | id ) ||  DC{y1}.( S{e2}.VM.(Idle | id ) | id )  || LC(r1_Send_Time){c1}
  -[1]->
  EU{x1,y1,e1}.id  ||  DC{y1}.( S{e2}.VM.(R{x1,c1}.(ID(i)  | Processing) | id ) | id  )  || LC(r1_Send_Time){c1} ;

fun react returnRequest_S1(i, s1_Process_Time) =
  EU{x1,y1,e1}.id  ||  DC{y1}.( S{e2}.VM.(R{x1,c1}.(ID(i)  | Processing) | S1 ) | id  )  || LC(s1_Process_Time){c1}
  -[1]->
  EU{x1,y1,e1}.( R{x1,c1}.( ID(i) | Result ) | id ) ||  DC{y1}.( S{e2}.VM.(Idle | S1 ) | id ) || LC(0){c1};

fun react returnRequest_S2(i, s2_Process_Time) =
  EU{x1,y1,e1}.id ||  DC{y1}.( S{e2}.VM.(R{x1,c1}.(ID(i)  | Processing) | S2 ) | id  ) || LC(s2_Process_Time){c1}
  -[1]->
  EU{x1,y1,e1}.( R{x1,c1}.( ID(i) | Result ) | id ) ||  DC{y1}.( S{e2}.VM.(Idle | S2 ) | id )  || LC(0){c1};

react done =
  EU{x1,x2,e1}.( R{x1,c1}.(Result | id ) | R{x2,c2}.(Result | id ) ) | EU{y1,y2,e1}.( R{y1,c3}.( Result | id ) | R{y2,c4}.(Result | id ) )
  -[1]->
  EU{x1,x2,e1}.( R{x1,c1}.(Stop | id ) | R{x2,c2}.(Stop | id ) ) | EU{y1,y2,e1}.( R{y1,c3}.( Stop | id ) | R{y2,c4}.(Stop | id ) ) ;


################# Initial State #################
big cloudSystem =
/x1/x2/y1/y2/e1/e2/c1/c2/c3/c4 (
 FrontEnd.(
   EU{x1,x2,e1}.( R{x1,c1}.(Wait | ID(1) ) | R{x2,c2}.(Wait | ID(2) ) ) | EU{y1,y2,e1}.( R{y1,c3}.( Wait | ID(3) ) | R{y2,c4}.(Wait | ID(4) ) )
)
|| BackEnd.(DC{e1}.( S{e2}.VM.(Idle | S1 ) | S{e2}.VM.(Idle | S2 )) )
|| LocalClock.( LC(0){c1} | LC(0){c2} |  LC(0){c3} | LC(0){c4} ) | GC(0)
);

################# Predicates #################
fun big request_Sent_to_S1_at(i, x) =
   S{e2}.VM.(R{x1,c1}.(ID(i) | Processing ) | S1 )
|| LC(x){c1};

fun big request_Return_at(i, x) =
R{x1,c1}.(ID(i) | Result ) || LC(x){c1};


begin abrs
  int i = {1, 2, 3, 4};
  int request1Clock={0,1,2,3,4,5,6,7,8};
  int request2Clock={0,1,2,3,4,5,6,7,8};
  int request3Clock={0,1,2,3,4,5,6,7,8};
  int request4Clock={0,1,2,3,4,5,6,7,8};
  int gc={0,1,2,3,4,5,6,7,8,9,10,11,12};

  int r1_Send_Time={1};
  int r1_Max_Send_Time={2};
  int r2_Send_Time={3};
  int r2_Max_Send_Time={4};
  int r3_Send_Time={4};
  int r3_Max_Send_Time={5};
  int r4_Send_Time={5};
  int r4_Max_Send_Time={6};

  int r1_Process_Time_S1={3};
  int r1_Process_Time_S2={4};
  int r2_Process_Time_S1={5};
  int r2_Process_Time_S2={6};
  int r3_Process_Time_S1={6};
  int r3_Process_Time_S2={7};
  int r4_Process_Time_S1={7};
  int r4_Process_Time_S2={8};

  int x ={0,1,2,3,4,5,6,7,8,9,10,11};


init cloudSystem;

  rules = [
        {done},
        {returnRequest_S1(1, r1_Process_Time_S1), returnRequest_S1(2, r2_Process_Time_S1), returnRequest_S1(3, r3_Process_Time_S1), returnRequest_S1(4, r4_Process_Time_S1),

        returnRequest_S2(1, r1_Process_Time_S2), returnRequest_S2(2, r2_Process_Time_S2), returnRequest_S2(3, r3_Process_Time_S2), returnRequest_S2(4, r4_Process_Time_S2)},

        {sendingRequest(1,r1_Max_Send_Time), sendingRequest(2,r2_Max_Send_Time), sendingRequest(3,r3_Max_Send_Time), sendingRequest(4, r4_Max_Send_Time)} ,

        {sendingRequest(1,r1_Send_Time), sendingRequest(2,r2_Send_Time), sendingRequest(3,r3_Send_Time),
        sendingRequest(4, r4_Send_Time), clock_advance(request1Clock,request2Clock,request3Clock, request4Clock, gc)}
   ];
```

```

actions=[
 send={sendingRequest},
 return = { returnRequest_S1 , returnRequest_S2 },
 tick = {clock_advance},
 stop = {done}
];

preds = {
request_Sent_to_S1_at(i, x), request_Return_at(i, x)};
end
```