Timed-Rebeca Schedulability and Deadlock-Freedom Analysis Using Floating-Time Transition System

Ehsan Khamespanah
School of Electrical and Computer Engineering, University of Tehran
e.khamespanah@ece.ut.ac.ir

Zeynab Sabahi Kaviani
School of Electrical and Computer Engineering, University of Tehran
z.sabahi@ece.ut.ac.ir

Ramtin Khosravi
School of Electrical and Computer Engineering, University of Tehran
r.khosravi@ut.ac.ir

Marjan Sirjani
School of Computer Science, Reykjavik University
marjan@ru.is

Mohammad-Javad Izadi
School of Electrical and Computer Engineering, University of Tehran
m.j.izadi@ece.ut.ac.ir

Abstract

"Timed-Rebeca" is an actor-based modeling language for modeling real-time reactive systems. Its high-level constructs make it more suitable for using it by software practitioners compared to timed-automata based alternatives. Currently, the verification of Timed-Rebeca models is done by converting into timed-automata and using UPPAAL toolset to analyze the model. However, state space explosion and time consumption are the major limitations of using the back-end timed automata model for verification. In this paper, we propose a new approach for direct schedulability checking and deadlock freedom verification of Timed-Rebeca models. The new approach exploits the key feature of Timed-Rebeca, which is encapsulation of concurrent elements. In the proposed method, each state stores the local time of each actor separately, avoiding the need for a global time in the state. This significantly decreases the size of the state space. We prove the bisimilarity of the generated transition system (called floating-time transition system) and the state space generated based on Timed-Rebeca semantics. Also, we provide experimental results showing that the new approach mitigates the state space explosion problem of the former method and allows model-checking of larger problems.

AGERE! 2012, October 21–22, 2012, Tucson, Arizona, USA.

Categories and Subject Descriptors D.2.4 [*SOFTWARE ENGINEERING*]: Software/Program Verification - Assertion checkers, Class invariants, Correctness proofs, Formal methods, Model checking, Programming by contract, Reliability, Statistical methods, Validation.

Keywords Actor model, Timed-Rebeca, Verification, Real-time systems, Schedulability, Deadlock

1. Introduction

In recent years, the modeling and verification of schedulability and deadlock freedom of component-based and distributed realtime systems has become very important [16]. Distributed and component-based systems consist of multiple cooperating components where the components are typically encapsulated subsystems or objects spread over a network, interacting using asynchronous communication. Providing quality of service guarantees—despite the ever-increasing complexity of distributed systems—has been and remains a grand challenge. Using formal methods, in general, and model-checking [8], in particular, has been advocated as a response to this challenge. Model checking tools exhaustively explore the state space of a system to make sure that a given property holds in all possible executions of the system. The properties preserve both functional and timing correctness of realtime distributed systems.

A well-established paradigm for modeling distributed and asynchronous component-based systems is the *Actor* model. This model was originally introduced by Hewitt [10] as an agent-based language and a mathematical model of concurrent computation. It treats *actors* as the universal primitives of concurrent computation [4, 12]. The asynchronous and nonblocking message-based communication and encapsulation of state and behavior make the actor model suitable for modeling of component-based software [11]. Each ac-

tor provides a certain number of services which can be requested by other actors by sending messages to the provider. Messages are put in the message buffer of the actor, then the actor takes the message and executes the requested service, possibly responding to some other components. There are some extension on the actor model for realtime systems like RT-synchronizer [20], realtime Creol [9], and Timed-Rebeca[3].

Reactive Objects Language, *Rebeca* [23], is an operational interpretation of the actor model with formal semantics, supported by model-checking tools. *Rebeca* is designed to bridge the gap between formal methods and software engineering. The formal semantics of *Rebeca* is a solid basis for its formal verification. Compositional and modular verification, abstraction, symmetry and partial-order reduction have been investigated for verifying Rebeca models. The theory underlying these verification methods is already established and is embodied in verification tools [15, 21–23]. With its simple, message-driven and object-based computational model, Java-like syntax, and a set of verification tools, *Rebeca* is an interesting and easy-to-learn model for practitioners.

Timed-Rebeca [3] has been proposed as an extension of *Rebeca* for modeling actor-based distributed and realtime systems. One of the first approaches in verifying Timed-Rebeca models, is suggested in [13]. In this method, the Timed-Rebeca model is translated into a set of timed automata, that was claimed to be collectively bisimilar to the original Timed-Rebeca model. The generated timed automata model, can then be verified using timed automata tools such as UPPAAL [7]. The drawback is that the resulting timed automata model generates a huge state space, which tends to grow exponentially as the model becomes larger. This is a result of the difference in the nature of the two models: while Timed-Rebeca uses asynchronous message passing , UPPAAL uses transition synchronisation as the only form of communication between the automata. Therefore, the mapping needs to use extra automata locations and transitions to simulate asynchrony. In addition, the translation algorithm uses a set of clocks to convert the absolute time model of Timed-Rebeca into a relative time model. The number of clocks used, which is dependent on the number of concurrent messages, highly affects the size of the state space and the model-checking time consumption.

There is the same drawback using *Realtime Maude* [19] and *Timed I/O Automata* [18] as back-end model checkers of Timed-Rebeca models. Realtime Maude is a timed logic language which has the ability to define a tick rule for time progress. The tick rule specifies the amount of the global time lapse for each step in model progress. So the synchronous tick progression causes unnecessary interleaving of independent behavior of components. Timed I/O automata is a basic mathematical framework to support description and analysis of timed computing systems [17]. The timed I/O automata formalism supports decomposing timed system descriptions to a number of automata. In particular, the framework includes a notion of external behavior for a timed I/O automaton, which captures its discrete interactions with its environment. A composition operation for timed I/O automata assigns an execution fragment to each action implying that a timed I/O automata does not block the passage of time. Such a fine-grained time progress causes unnecessary interleaving of concurrent independent actions which increases the size of the state space.

In this work, we introduce the notion of *floating-time transition system* for tackling the state space explosion problem for Timed-Rebeca. Contrary to timed automata and timed I/O automata where the modeler resets the clocks explicitly to avoid unbounded time progress, the progress of time in Timed-Rebeca models will automatically become bounded by using floating-time transition system. Our technique is detecting the recurrent patterns of behaviors while building the transition system of the model, and allowing different local times for each actor in the same state. This is achieved by having relative time specifiers in Timed-Rebeca models.

In all other described methods, each state has a consistent time interval specifier. Therefore, at each state the global time specification of the state is defined. In contrast, the new approach is developed based on the key ingredient of Timed-Rebeca models, which is encapsulation of concurrent elements. Each state stores the local times of its concurrent elements. Therefore, there is no need for global time in the state for deadlock freedom and schedulability analysis. The local times of different actors in a state can be different because each element has its own local message and time management. This property significantly decreases the size of the state space as will be shown in the experimental results. In this paper, we define the formal definition of the floating-timed transition system and a bisimulation relation between this transition system and the transition system derived form SOS (Structural Operational Semantics) of Timed-Rebeca[3]. Based on this bisimilarity, it can be concluded that deadlock freedom and schedulability analysis of SOS and floating-time transition systems have similar results.

Motivation and Contribution. The contributions of this paper can be summarized as follows:

- Introducing the notion of floating-time transition system as an abstract way for state space generation of Timed-Rebeca models
- Proving the bisimilarity of the floating-time transition system and the SOS transition system of Timed-Rebeca models by introducing a behavioral equivalency of timed systems states as time-shift equivalency relation
- Implementing a tool for schedulability and deadlock-freedom analysis based on the proposed techniques

– Providing experimental results and measuring the efficiency of our technique by means of a number of case studies

Roadmap. The rest of this paper is structured as follows. The next section gives some background about the Timed-Rebeca modeling language, its operational semantics, and its translator to the timed automata. Section 3 defines the concept of floating-time transition system. Section 4 defines the schedulability and deadlock analysis of the floating-time transition system, proving the existence of bisimulation relation between SOS rules and the floating-time transition system. In Sections 5 and 6, we present the implementation issues of state space generation algorithm and report the experimental results respectively. The concluding remarks are presented in Section 7.

2. Preliminaries

2.1 Timed-Rebeca

Timed-Rebeca [3] has been proposed as an extension of *Rebeca* [23, 24] for modeling actor-based distributed and realtime systems. A Timed-Rebeca model consists of the definition of *reactive classes* and the instantiation part which is called *main*. The main part defines instances of reactive classes, called *rebec*s. The reactive class comprises three parts: *known rebecs*, *state variables*, and *message server* definitions.

The known rebecs of a reactive class are the destination rebecs of the messages which may be sent by the instances of the reactive class. The internal state of a reactive class is represented by the valuation of its state variables. Because of the encapsulation of *actor*s, the state variables of an actor cannot be directly accessed by other actors. The behavior of the instances of a reactive class is determined by the definitions of its message servers. In Rebeca, communication takes place by asynchronous message passing, which is non-blocking for both sender and receiver. The sender rebec sends a message to the receiver rebec and continues its work. The message is immediately put in the message bag of the receiver until its release time. It is then taken from the bag and the execution of the corresponding message server is started. Execution of a message server body takes place non-preemptively. Each message server has a name, a (possibly empty) list of parameters, and the message server body which includes a number of statements. The statements may be assignments, sending of messages, selections, and delays. We do not consider dynamic rebec creation or reference passing (dynamic topology). Note that there is no explicit receive statement in Rebeca. Time progress is modeled by delays and the time quantifiers of messages (Message release time and deadline). We illustrate this with an example. Figure 1 shows the Timed-Rebeca model of a ticket service system. The model consists of three reactive classes: *TicketService*, *Agent*, and *Customer*. *Customer* sends the "ticket issue" message to *Agent* and *Agent* forwards the issue to *TicketService*. *TicketService* rebec replies to *Agent* by sending a "ticket issued" message and *Agent* responds to *Customer* by sending the issued ticket identifier. As shown in line 12 of the model, issuing the ticket takes three time units (based on the configuration parameters, the issueDelay initial value equals to three). In addition, line 24 shows that *Agent* waits for five time units for *TicketService* to take the requested message and starts serving it.

The behavior of a Rebeca model is defined as the parallel execution of the released messages of the rebecs. At the initialization state, a number of rebecs are created statically, and an "*initial*" message is implicitly put in their bags. The release times of the initial messages are zero. The execution of the model continues as rebecs change the values of their state variables and send messages to each other.

2.1.1 Timed-Rebeca Structural Operational Semantics

In this section we provide an overview of the SOS semantics of Timed-Rebeca which has been proposed in [3].

Timed-Rebeca states are pairs (Env, B), where Env is a finite set of environments and B is a bag of messages. For each rebec r of the model there exists $\sigma_r \in Env$ which stores information about the actor, including the values of its state variables and local time, as well as structural characteristics like the body of the message servers. The bag B contains an unordered collection of all the messages of all rebecs. Each message is a tuple of the form $(r_i, m(v), r_j, TT, DL)$. Intuitively, such a tuple says that at time TT (time tag), the sender r_j sent the message to the rebec r_i requesting it to execute its message server m with actual parameters v. Moreover this message expires at time DL [3]. Note that DL specifies the time that the message has to be released, i.e., the messages server has to start its execution..

The system transition relation $\rightarrow$ is defined by the rule *scheduler* of Figure 2 where the condition C is defined as follows: σ_{r_i} is not contained in Env, $\sigma_{r_i}(rtime) \leq DL$, $TT \leq min(B)$, and $(r_i, m(\overline{v}), r_j, TT, DL) \notin B$. The scheduler rule allows the system to progress by picking up messages from the bag and executing the corresponding methods. The third side condition of the rule, namely $\sigma_{r_i}(rtime) \leq DL$, checks whether the selected message carries an expired deadline, in which case the condition is not satisfied and the message cannot be picked. The last side condition is the predicate $TT \leq min(B)$, which shows that the time tag TT of the selected message is the smallest time tag of all the messages (for all the rebecs r_i) in the bag B [3].

The τ transition shows the execution of the message server of the rebec r_i and its effects formally defined in [3]. Intuitively, execution of a message server means carrying out its body statements atomically. The execution may change the environment of the rebec r_i by assigning new values to its state variables, modifying the *now* value of r_i, or

$$\frac{(\sigma_{r_i}(m), \sigma_{r_i}[rtime = max(TT, \sigma_{r_i}(now))], [\overline{arg} = \overline{v}], sender = r_j], Env, B) \xrightarrow{\tau} (\sigma'_{r_i}, Env', B')}{(\{\sigma_{r_i}\} \cup Env, \{(r_i, m(\overline{v}), r_j, TT, DL)\} \cup B) \rightarrow (\{\sigma'_{r_i}\} \cup Env', B')} C$$

Figure 2. Timed-Rebeca system transition relation scheduler sos rule

changing the bag by sending messages to other rebecs. The new message is put in the bag with its time tag and deadline. Time tag is its relative receive time which is computed by the value of *after* statement. The $\sigma_{r_i}(now)$ (current time of rebec) may be modified if the body of message contains the timing statement *delay*. The current time of the rebec increases by the value of the *delay* statement.

2.2 Model Checking Timed-Rebeca using UPPAAL

Mapping the Timed-Rebeca model to timed automata [5] and model-checking the resulting timed automata is an approach which has been suggested in [13]. In this approach, the author generates three timed automata for each rebec. These three automata model the behavior of the message servers, the timed-bag, and the "after" usage in the Timed-Rebeca model. Such a mapping does not seem very straightforward mainly because in Timed-Rebeca message passing is asynchronous, while timed automata models have a synchronous messaging mechanism. There were also some other obstacles, which will be explained in the following lines, together with the suggested solutions.

The author considers a single timed automaton per rebec, called rebec timed automaton. This timed automaton has (as its internal variable) an array that models the message bag. To implement sending messages, the underlying timed automata synchronization mechanism over channels is used. This requires the receiving timed automaton to always be ready to accept messages (as specified in the semantics of Timed-Rebeca: messages are instantaneously received in the message bag of the receiver). For this, it is needed to have transitions on every location in the rebec timed automata, which accepts the message synchronously and puts it in the message bag of the receiving rebec.

Another solution would be to consider another timed automaton per rebec to model its message bag. This timed automaton, called the rebec's Inbox timed automaton, would then always accept messages asynchronously, regardless of the state of the corresponding rebec timed automaton, and then deliver it, upon the rebec timed automaton's request. The Inbox rebec is responsible to handle message activation time and deadlines. Because of its simplicity and better readability the author has chosen the latter approach. Early comparisons between these approaches did not show significant performance difference. Figure 3 shows the timed automaton of rebec's Inbox. As depicted in Figure 3, rebec's Inbox automaton inserts the incoming messages of the owner rebec, discards the messages with passed deadlines, and extracts the messages from Inbox and delivers them.

The rebec timed automaton, will then need to only model the behavior of the rebec itself, as specified by its message servers and state variables' valuations. State variables are converted to rebec timed automaton variables. For the message servers, first consider a timed automaton which simulates the exact behavior of the message server. For simple statements like conditionals, loops, assignments, etc, the mapping is straightforward. Delays are also trivially converted, by the use of one clock, and addition of a location and transition guards to the timed automaton. Figure 4 shows implementations of these statements in timed automata. The mapping for message sending statements will be described shortly.

After deriving the timed automaton equivalent to each of the message servers, all these timed automata are integrated into a single timed automaton and the constructs needed for receiving the next message from the Inbox timed automaton are added. Then the timed automaton will find out which message it has received, and direct the execution to the corresponding message server. Upon completion of the execution of the message server, the rebec timed automaton will requests for the next message, and responds to that message. This completes the reactive behavior of the rebec.

To implement sending of messages, first the exact time constructs of the timed automaton should be specified. In Timed-Rebeca, each rebec has an internal clock, which shows the time elapsed since the creation of the rebec. This specifies an absolute model of time, which cannot be implemented in timed automata, because it makes clock values to grow unboundedly. Therefore, this absolute time model should be converted to relative time. The only time-related constructs in Timed-Rebeca are delay statements and message sending statements. The case for delays is studied before. For message sending, instead of giving timestamps to messages, the author attached one clock to the message. The clock is extracted from a pool of clocks. This clock is used when checking activation times and deadlines and is returned to the pool, when the message is delivered to the rebec timed automaton for execution.

In addition to the formal specification of the mapping algorithm, the author has proved the existance of an equivalence between the resulting timed automata and the structural operational semantics of Timed-Rebeca and has developed a tool for automatic mapping of the Timed-Rebeca models to timed automata. The parallel composition of the resulting timed automata and the schedulability analysis of the model is done using the UPPAAL toolset[7].

```
reactiveclass TicketService {
  knownrebecs {
    Agent a;
  }
  statevars {
    int issueDelay;
  }
  msgsrv initial(int myDelay) {
    issueDelay = myDelay;
  }
  msgsrv requestTicket() {
    delay(issueDelay);
    a.ticketIssued(1);
  }
}
reactiveclass Agent {
  knownrebecs {
    TicketService ts;
    Customer c;
  }
  msgsrv requestTicket() {
    ts.requestTicket()
      deadline(5);
  }
  msgsrv ticketIssued(byte id) {
    c.ticketIssued(id);
  }
}
reactiveclass Customer {
  knownrebecs {
    Agent a;
  }
  msgsrv initial() {
    self.try();
  }
  msgsrv try() {
    a.requestTicket();
  }
  msgsrv ticketIssued(byte id) {
    self.try() after(30);
  }
}
main {
  Agent a(ts, c):();
  TicketService ts(a):(3);
  Customer c(a):();
}
```

Figure 1. The Timed-Rebeca model of ticket service system.

A major limiting factor in using UPPAAL and applying the proposed method to practical systems is the generation of huge state space. Generating the state space of large-scale practical timed systems undoubtedly results in state spaces that cannot fit in the memory of a computer. In addition to memory limitation, the model-checking time consumption increases rapidly and makes the model-checking impossible. We will come back to these limitations in Section 6.

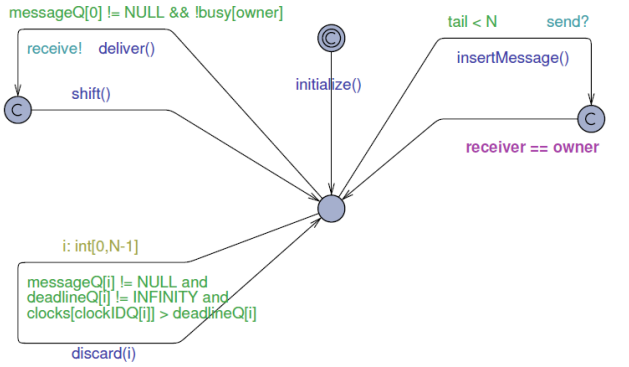
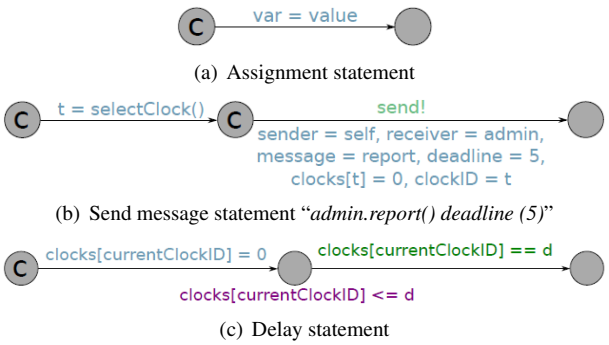


Figure 3. Rebec's Inbox automaton



(a) Assignment statement

(b) Send message statement "*admin.report() deadline (5)*"

(c) Delay statement

Figure 4. Implementation of three different Timed-Rebeca statements in timed automata

3. Floating-Time Transition System of Timed-Rebeca

In this section, we describe the floating-time transition system of Timed-Rebeca models which is used for schedulability and deadlock-freedom analysis.

DEFINITION 1 (Rebecs of a Timed-Rebeca Model). *For a Timed-Rebeca model Reb, the function $O(Reb)$ returns all rebec instances in the model.*

The state of a rebec in model Reb consists of the values of its state variables, its local time, and its message bag. For a Timed-Rebeca model Reb the collection of the states of all the rebecs of $O(Reb)$ is called a Timed-Rebeca state.

DEFINITION 2 (Timed-Rebeca State). *A state of a Timed-Rebeca model is a tuple $s = \prod_{r_i \in O(Reb)} \text{state}(r_i)$, where $\text{state}(r_i)$ is the current state of rebec r_i. The functions $\text{statevars}(s, r_i)$, $\text{bag}(s, r_i)$, and $\text{now}(s, r_i)$ return the state variables valuation function, the message bag content, and the current time of rebec r_i in state s, respectively.*

S_0	
a	State vars:
	Message Bag: $[(initial, 0, \infty)]$
	Now: 0
ts	State vars: issueDelay=?
	Message Bag: $[(initial, 0, \infty)]$
	Now: 0
c	State vars:
	Message Bag: $[(initial, 0, \infty)]$
	Now: 0

I The initial state

S_{15}	
a	State vars:
	Message Bag: [ ]
	Now: 6
ts	State vars: issueDelay=3
	Message Bag: [ ]
	Now: 6
c	State vars:
	Message Bag: $[(ts.ticketissued, 6, \infty)]$
	Now: 0

II An intermediate state

Figure 5. The initial state and one of other states of the model of ticket service system which has been depicted in Figure 1. The receiver of each message is shown in the left-most column (as a, ts, c). Each message is shown as: (sender.message-server-name (list of actual parameters), arrival-time, deadline)

Two different states of the Timed-Rebeca model of Figure 1 are depicted in Figure 5. The state in Figure 5I is an initial state with its rebecs' "now" time set to zero and the "initial" message put in their message bags. The sender of the initial messages are omitted in Figure 5I because the initial messages have no specific sender. Figure 5II depicts one of the intermediate states of the model. In Figure 5II rebec c has a message from rebec ts for time 6 which has no argument. As shown in Figure 5II, there is no guarantee on the equality of the local times of rebecs of a state, so we call it "Floating Time State (FTS)". To ease the reading of the paper, we use "state" instead of FTS in the following.

As shown in definition 2 the state contains the rebecs' message bags. The message bag of rebec is an unordered collection of messages which is structured as defined in the following definition.

DEFINITION 3 (Rebec Message Bag). *A message* $\mathrm{tmsg} = (\mathrm{msgsig}, \mathrm{arrival}, \mathrm{deadline})$ *is a message where* msgsig *is the message content,* arrival *is the arrival time of the message (which is computed based on the value of "after" of send message statement in a Timed-Rebeca model), and* deadline *is the deadline of the message based on the rebec local time. The message msgsig consists of message name, the sender, the receiver, and its actual parameters. For* $\mathrm{tmsg} \in \mathrm{bag}(s, r_i)$ *the functions* $\mathrm{msgsig}(\mathrm{tmsg})$, $\mathrm{arrival}(\mathrm{tmsg})$, *and* $\mathrm{deadline}(\mathrm{tmsg})$ *return the msgsig, arrival, and deadline of the message* tmsg.

The *release time* of a message is the earliest time in which it can start its execution. It depends on the arrival time of the message and the current time of the rebec. The release time of a received message can be later than its arrival time, because upon arrival the receiving rebec may still be busy executing another message server. This happens because message servers execute non-preemptively. Therefore, the execution of a message may delay the execution of another enabled message of a rebec.

DEFINITION 4 (Message Release Time). *The message release time of* $\mathrm{tmsg} \in \mathrm{bag}(s, r_i)$ *is defined as* $\max\{\mathrm{now}(s, r_i), \mathrm{arrival}(\mathrm{tmsg})\}$, *denoted by* $\mathrm{releasetime}(\mathrm{tmsg})$.

The next enabled messages of each rebec is defined based on the Earliest-Arrival-Time-First policy of Timed-Rebeca using the messages' arrival times. The set of enabled messages are called *enabled messages* which is a set of messages which should be executed before other messages of r_i.

DEFINITION 5 (Rebec Enabled Messages). *For state* s *and rebec* $r_i \in O(\mathrm{Reb})$, $\mathrm{enabledmessages}(s, r_i) = \{\mathrm{tmsg} \in \mathrm{bag}(s, r_i) | \forall \mathrm{tmsg}' \in \mathrm{bag}(s, r_i) \cdot \mathrm{arrival}(\mathrm{tmsg}) \leq \mathrm{arrival}(\mathrm{tmsg}')\}$.

DEFINITION 6 (Rebec Next Message Release Time). *The release time of the currently enabled messages of rebecs* r_i *in state* s *is defined as* $\mathrm{NMRT}(s, r_i) = \mathrm{releasetime}(\mathrm{enabledmessages}(s, r_i))$.

Based on definitions 1 to 6, a new equivalence relation between states is defined. The suggested equivalence relation helps to have a bounded state space and avoid infinite state space which occurs because of the time progress of the model. For an informal and simplified description, note that the time expressions used in Timed-Rebeca delay and message-sending statements are relative times. Therefore, for two states s and s' which only differ in the local time of rebecs, shifting the local time and messages time qualifiers of all the rebecs of state s' backwards to make it equal to s, does not affect its behavior. It only affects the arrival and deadline times of the sent messages, so the behavior of the model after state s' is the same as s and there is no need for state generation after reaching s'. We illustrate this with an example. Consider Figure 6 presenting three different states of the Timed-Rebeca model of the ticket service system of Figure 1.

Assume that the model is in the state shown in Figure 6I. Based on the bag of the rebecs *ts*, *c*, and *a*, $\mathrm{enabledmessages}(S_{20}, a) = \{\}$, $\mathrm{enabledmessages}(S_{20}, ts) = \{\}$, and for rebec c $\mathrm{enabledmessages}(S_{20}, c) = \{[\mathrm{ticketIssued}, 36, \infty]\}$. Therefore, only rebec *c* has an enabled message whose execution results in S_{21}, shown in Figure 6II.
Here, shifting the time of S_{21} by 33 units, makes it equal to

S_{20}	
a	State vars: Message Bag:[ ] Now: 36
ts	State vars: issueDelay=3 Message Bag:[ ] Now: 36
c	State vars: Message Bag:[$(c.ticketIssued, 36, \infty)$] Now: 36

I State number 20

S_{21}	
a	State vars: Message Bag:[ ] Now: 36
ts	State vars: issueDelay=3 Message Bag:[ ] Now: 36
c	State vars: Message Bag:[$(c.try, 66, \infty)$] Now: 36

II State number 21

S_{16}	
a	State vars: Message Bag:[ ] Now: 3
ts	State vars: issueDelay=3 Message Bag:[ ] Now: 3
c	State vars: Message Bag:[$(c.try, 33, \infty)$] Now: 3

III State number 16

Figure 6. Three different states of the Timed-Rebeca model of ticket service system

S_{16}, so S_{21} and S_{16} are shift equivalent by shifting 33 units, denoted by $S_{21} \cong_{33} S_{16}$.

DEFINITION 7 (Rebec State Shift Equivalence). *Two states s and s' are shift equivalent with respect to rebec r_i, if* $\text{statevars}(s, r_i) = \text{statevars}(s', r_i)$ *and one of the following conditions hold:*

- $\text{now}(s', r_i) = \text{now}(s, r_i)$ *and* $\text{bag}(s', r_i) = \text{bag}(s, r_i)$. *In this case s' and s are zero-time shift equivalent with respect to r_i, denoted by $s' \cong_{i,0} s$.*
- *There exists an integer number t such that* $\text{now}(s', r_i) = \text{now}(s, r_i) + t$ *and there is a bijective relation $\leftrightarrow$ between* $\text{bag}(s', r_i)$ *and* $\text{bag}(s, r_i)$ *where for* $\text{tmsg}' \in \text{bag}(s', r_i)$ *and* $\text{tmsg} \in \text{bag}(s, r_i)$ $\text{tmsg}' \leftrightarrow \text{tmsg}$ iff $\text{msgsig}(\text{tmsg}') = \text{msgsig}(\text{tmsg}) \wedge \text{arrival}(\text{tmsg}') = \text{arrival}(\text{tmsg}) + t \wedge \text{deadline}(\text{tmsg}') = \text{deadline}(\text{tmsg}) + t$. *In this case, the two states are "by t unit(s)" shift equivalent with respect to r_i, denoted by $s' \cong_{i,t} s$. Note that the time specifiers of Timed-Rebeca are integer numbers, hence t is a member of natural numbers.*

For two states s and s' shift equivalence is defined as $\exists t \in \text{N} \cdot \forall r_i \in O(Reb) \cdot s' \cong_{i,t} s$, *denoted by $s' \cong_t s$. Note that the time shift preserves the relative difference of the rebec now and its bag's message specifiers.*

Now the Timed-Rebeca floating-time transition system can be defined based on the notion of shift equivalence. The floating-time transition system is a transition system similar to Figure 7 which depicts a floating-time transition system of the ticket-service model. To make the transition system easy to understand, transition labels from state S_0 to state S_{15} are omitted. As shown in Figure 7, a transition label is a pair consisting of the executed message and the time shift. The time shifts of all the transitions of Figure 7 are zero except the transition from S_{20} to S_{16}, because of the equivalence relation defined in Definition 7.

DEFINITION 8 (Timed-Rebeca Floating-Time TS). *A* Timed-Rebeca Floating-Time Transition System (FTTS) *is a labeled transition system* $\text{FTTS}(Reb) = (S, s_0, Act, \hookrightarrow, AP, L)$, *where:*

- *S is a set of states.*
- *$s_0 \in S$ is an initial state.*
- *Act is a set of actions. Each action is a pair of message and time shift.*
- *$\hookrightarrow \subseteq S \times \text{Act} \times S$ is a set of transition relations.* $\forall s, s' \in S, (s, (tmsg, t), s') \in \hookrightarrow$ *iff there exists* $r_i \in O(Reb), \text{tmsg} \in \text{enabledmessages}(s, r_i)$ *such that the execution of* tmsg *results in a state s'' where* $s'' \cong_t s'$ *and* $\forall r_j \in O(Reb) \cdot \forall \text{tmsg}' \in \text{enabledmessages}(s, r_j) \cdot \text{arrival}(\text{tmsg}) \leq \text{arrival}(\text{tmsg}')$.
  The execution of message tmsg conforms to the Timed-Rebeca semantic SOS rule which sets $now(s, r_i)$ *to* $\text{NMRT}(s, r_i)$ *before starting execution of the first statement of tmsg, modifies the rebec's state variables, sends some messages to other rebecs, and change the local time of rebec as described in Section 2.1.1.*
  There is a non-deterministic choice in those states with more than one enabled message.
- *AP=*$\{\text{Deadlock}, \text{DeadlineMissed}\}$ *is a set of atomic propositions.*
- *$L : S \to 2^{AP}$ is a labeling function defined over the set of states as follows:*
  - *Deadlock* $\in L(s)$ iff $\nexists s', act \cdot (s, act, s') \in \hookrightarrow$.
  - *DeadlineMissed* $\in L(s)$ *iff there exists a state s and a rebec* $r_i \in O(Reb)$ *such that* $\text{NMRT}(s, r_i) > \text{now}(s, r_i)$.

4. Transition System Schedulability and Deadlock-Freedom Analysis

A Timed-Rebeca model is schedulable if the deadline of no message is missed while executing the model. In other

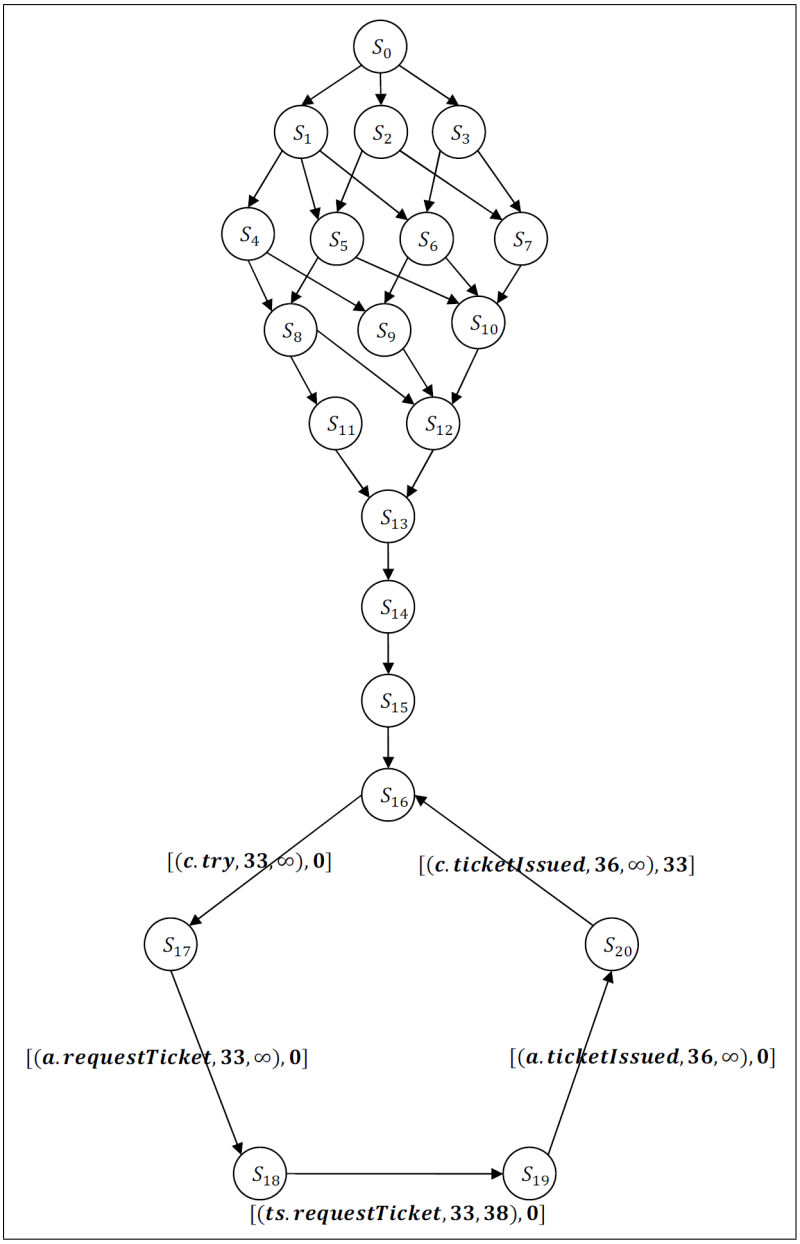


Figure 7. Floating-Time transition system of the Timed-Rebeca model of Figure 1.

words, there is no reachable state such that the value of "now" of a rebec is greater than the deadline of any message in its bag. Because of the progress of time in Timed-Rebeca semantics, applying the SOS rules on a model results in an unbounded state space which makes the schedulability analysis impossible. However, the bisimilarity of SOS-based transition system and the floating-time state space of a model should be proved. Since the elements of SOS-based transition system and floating-time transition system are syntactically different, in the first step, the following functions should be defined to extract the elements of a state of the SOS-based transition system.

DEFINITION 9 (Functions on SOS Rules Based TS). *Consider an unbounded transition system derived from applying the Timed-Rebeca SOS rules [3] on Reb such as* $\text{SOSTS} = (S, s_0, Act, \rightarrow, AP, L)$*, a state* $s = (Env, B) \in S$*,* $r_i \in O(Reb)$*, and* $\sigma_{r_i} \in Env$*.*

- *the function* $\text{bag}(s, r_i)$ *returns* $\{(r_t, m(\overline{v}), r_s, TT, DL) \in B | r_s = r_i\}$ *,i.e., the bag of messages of rebec* r_i *in state* s*.*
- *the function* $\text{now}(s, r_i)$ *returns* $\sigma_{r_i}(now)$*, i.e., the "now" of rebec* r_i *in state* s*.*

- *the function* $\text{statevars}(s, r_i)$ *returns the valuation function of the state variables of rebec* r_i *in state* s *which is extracted from* σ_{r_i}.

Here, $AP = \{\text{Deadlock}, \text{DeadlineMissed}\}$ and L for a state $s = (Env, B) \in S$ is defined as follows:

- *Deadlock* $\in L(s)$ iff $B = \emptyset$.
- *DeadlineMissed* $\in L(s)$ iff for some $(r_i, m(v), r_j, TT, DL) \in B$ condition $max\{\sigma_{r_i}(now), TT\} > DL$ holds.

Before presenting the bisimilarity theorem, it should be remarked that shifting the time of a state preserves its enabled messages. Consider two states $s_1, s_2 \in S'$ where $s_1 \cong_t s_2$ for some $t \in \mathbb{N}$ and a rebec $r_i \in O(Reb)$. Based on Definition 7, $\forall \text{tmsg}_1 \in \text{bag}(s_1, r_i)$ there exists $\text{tmsg}_2 \in \text{bag}(s_2, r_i)$ such that $\text{arrival}(\text{tmsg}_1) + t = \text{arrival}(\text{tmsg}_2) \wedge \text{deadline}(\text{tmsg}_1) + t = \text{deadline}(\text{tmsg}_2) \wedge \text{msgsig}(\text{tmsg}_1) = \text{msgsig}(\text{tmsg}_2)$. So in case of $\text{tmsg}_1 \in \text{enabledmessages}(s_1, r_i)$ and $\text{tmsg}_2 \notin \text{enabledmessages}(s_2, r_i)$ there exists $\text{tmsg}'_2 \in \text{enabledmessages}(s_2, r_i)$ such that the $\text{arrival}(\text{tmsg}'_2) < \text{arrival}(\text{tmsg}_2)$. Here, based on shift equivalence relation, there exists $\text{tmsg}'_1 \in \text{bag}(s_1, r_i)$ such that $\text{arrival}(\text{tmsg}'_1) + t = \text{arrival}(\text{tmsg}'_2) \wedge \text{deadline}(\text{tmsg}'_1) + t = \text{deadline}(\text{tmsg}'_2) \wedge \text{msgsig}(\text{tmsg}'_1) = \text{msgsig}(\text{tmsg}'_2)$. So there exists unequality $\text{arrival}(\text{tmsg}'_1) < \text{arrival}(\text{tmsg}_1)$ which contradicts the assumption of $\text{tmsg}_1 \in \text{enabledmessages}(s_1, r_i)$.

THEOREM 1. *For a Timed-Rebeca model Reb, an* $\text{SOSTS} = (S, s_0, act, \rightarrow, AP, L)$ *which is an unbounded transition system derived from applying the Timed-Rebeca SOS rules [3] on Reb, is bisimilar to the Timed-Rebeca floating-time transition system* $\text{FTTS} = (S', s'_0, act', \hookrightarrow, AP', L')$ *of model Reb.*

PROOF 1. *Based on the renaming function and the Timed-Rebeca semantics, a bisimulation binary relation* $\mathcal{R} \subseteq S \times S'$ *is defined such that for two states* $s \in S$ *and* $s' \in S'$, $(s, s') \in \mathcal{R}$ *iff* $\exists s'' \in S', \exists t \in \mathbb{N} \cup \{0\} \cdot s'' \cong_t s'$ *and* $\forall r_i \in O(Reb) \cdot \text{bag}(s, r_i) = \text{bag}(s'', r_i) \wedge \text{statevars}(s, r_i) = \text{statevars}(s'', r_i) \wedge \text{now}(s, r_i) = \text{now}(s'', r_i)$.

Now it must be shown that for two states $s \in S$ *and* $s' \in S'$, *and* $(s, s') \in \mathcal{R}$, *the following bisimulation transfer conditions are satisfied [6].*

I. If $q \in \text{Post}(s)$ *then there exists* $q' \in \text{Post}(s')$ *with* $(q, q') \in \mathcal{R}$

II. If $q' \in \text{Post}(s')$ *then there exists* $q \in \text{Post}(s)$ *with* $(q, q') \in \mathcal{R}$

III. $L(s) = L'(s')$

In the above conditions Post(s) denotes all successor states of a state s according to the transition relation (for both transition systems). As mentioned in Definition 8, the execution of a message conforms to the Timed-Rebeca SOS rules, so the execution of a message has the same effect on both SOS-based transition system and floating-time transition system. Therefore, conditions I and II hold because as $(s, s') \in \mathcal{R}$ *they have the same enabled messages and the execution of enabled messages in both transition systems have the same effect on the values of state variables and the bags of rebecs.*

The third condition holds for s *and* s' *because* $AP = AP' = \{\text{Deadlock}, \text{DeadlineMissed}\}$ *and two following conditions hold:*

- *if* $\text{Deadlock} \in L(s)$ *then* $B = \emptyset$. *As* $(s, s') \in \mathcal{R}$, $\forall r_i \in O(reb) \cdot \text{bag}(s', r_i) = \emptyset$. *Therefore,* $\text{Deadlock} \in L'(s')$, *and vice versa.*
- *if* $\text{DeadlineMissed} \in L(s)$ *then* $(r_i, m(v), r_j, TT, DL) \in B$ *exists such that* $max\{\sigma_{r_i}(now), TT\} > DL$. *As the bag of the rebecs in* s *and* s' *are same (because* $(s, s') \in \mathcal{R}$*), the corresponding message* tmsg *in the bag of* r_i *in* s' *should exist such that by* $t \in \mathbb{N} \cup \{0\}$ *units time shift, its time quantifiers are the same as* TT *and* DL *and* $\text{now}(s, r_i) = \text{now}(s', r_i) + t$. *Therefore,* $\text{releasetime}(tmsg) > \text{deadline}(tmsg)$, *so* $\text{DeadlineMissed} \in L'(s')$, *and vice versa.*

COROLLARY 1 (Timed-Rebeca Model Analysis). *For the Timed-Rebeca model Reb both schedulability and deadlock-freedom can be examined using its floating-time state space.* $\text{SOSTS}(\text{Reb})$ *has a deadline missed message iff there is a state* s *in* FTTS *labeled DeadlineMissed and* $\text{SOSTS}(\text{Reb})$ *has a deadlock state iff there is a state* s *in* FTTS *labeled Deadlock.*

5. Implementation

Rebeca comes equipped with an on-the-fly explicit-state LTL model-checking engine called Modere [14]. Modere uses both the nested DFS and BFS search algorithms to explore the state space. To generate state space based on semantics of floating-time transition system and its required analyzer, Modere's BFS search algorithm has been extended to support Timed-Rebeca models, incorporating our novel shift equivalent states detection.

5.1 Rebeca BFS State Space Generation Algorithm

The BFS exploration algorithm, creates and explores the transition system in a level-by-level fashion. In the first step of the BFS algorithm, the initial state of the Rebeca state space is stored and marked as visited. Then, for each level, the successors of the states are generated by applying the successor function to the states; when there are no unexplored states in the next level, the algorithm terminates. For specification of the successor function and its formal semantics refer to [23].

The BFS state space generation algorithm can be implemented using two queues to manage states of each level. The first queue stores the current level states (CLQ) and the second one stores the successors of the CLQ states. The latter queue is called the next level queue (NLQ). In each iteration,

```
1  BFS-STATE-SPACE-GENERATOR ()
2    CLQ ← ∅
3    NLQ ← ∅
4    Visited ← ∅
5    ENQUEUE (CLQ, initState)
6    while CLQ ≠ ∅ do
7      for each state S ∈ CLQ do
8        NewStates ← SUCCESSOR(S)
9        for each State N ∈ NewStates do
10         if N ∉ Visited
11           then PUT(Visited, N)
12                PUT(LocalHashTable, N)
13                ENQUEUE(NLQ, N)
14         fi
15       od
16     od
17     swap(CLQ, NLQ)
18     NLQ ← ∅
19   od
```

Figure 8. Modere BFS state space generation pseudo code.

the unexplored states of the CLQ are dequeued and their unvisited successors are generated. When all states of the CLQ are dequeued, the content of the NLQ is moved to the CLQ and the algorithm continues until NLQ is empty, i.e., all successors of the states in the CLQ are visited before. Figure 8 shows a pseudo code of the algorithm.

5.2 Timed-Rebeca BFS State Space Generation Algorithm

From the implementation view, the major differences between Rebeca and Timed-Rebeca transition system generation algorithms are in the state structure. A list of time bundles is attached to each state to store the time specification of states – for each rebec its *now*, and for all the messages in the queues of each rebec the *deadline* and *after* specifiers. Therefore, state S with for example two time bundles in its time bundles list represents two different states which their rebecs state variables valuations and message queue content are the same, however the time specifiers are different. Therefore, the approach is to split state information into time-invariant (the valuation of variables and message queues) and time-dependent parts. Multiple time-dependent parts, which we call time bundles, associate with one time-invariant part. In this way, checking for time-shift equivalence of states can be done efficiently. Based on this structure, states shift equivalence checking has been reduced to the $O(|time\ bundles|)$ (the size of time bundles) search algorithm which has been shown in Figure 9 lines 17 to 29.

6. Experimental Results

Model-checking execution time and state space size of Timed-Rebeca models based on timed automata transition system and floating-time transition system are compared using three different examples. The examples are *Distributed Sensor Network*, simplified version of *Slotted ALOHA Protocol*, and *Ticket Service*. The test platform is a HP DL-386

```
1  BFS-STATE-SPACE-GENERATOR ()
2    CLQ ← ∅
3    NLQ ← ∅
4    Visited ← ∅
5    ENQUEUE (CLQ, initState)
6    while CLQ ≠ ∅ do
7      for each state S ∈ CLQ do
8        NewStates ← SUCCESSOR(S)
9        for each State N ∈ NewStates do
10         if N ∉ Visited
11           then PUT(Visited, N)
12                PUT(LocalHashTable, N)
13                ENQUEUE(NLQ, N)
14                CHECK-FOR-DEADLINE-MISSED(N)
15                CHECK-FOR-DEADLOCK(N)
16           else
17             SL ← GET(LocalHashTable, N)
18             for each bundle ∈ TIME-BUNDLES(SL) do
19               tb = TIME-BUNDLE(N)
20               if tb = bundle
21                 then
22                   // N ≅0 SL, so discard visited state
23                 elseif tb - t = bundle
24                   // N ≅t SL, so discard visited state
25                 else
26                   ADD-BUNDLE(N, tb)
27                   ENQUEUE(NLQ, N)
28               fi
29             od
30         fi
31       od
32     od
33     swap(CLQ, NLQ)
34     NLQ ← ∅
35   od
```

Figure 9. Timed-Rebeca BFS state space generation pseudo code.

G7 server with 4 CPUs of 2.80GHz Intel Xeon and 16GB of RAM storage running Ubuntu 12.04 operating system.

Sensor Network The distributed sensor network model is a model of a set of sensors which measure the toxic gas level of the environment. Upon sensing a dangerous level of gas, the sensors alarm the scientist who is working there to escape, or alternatively send a rescue team to save him.
There are four different reactive classes *Sensor*, *Admin*, *Scientist*, and *Rescue* in the model. *Sensor* rebecs send the measured gas level value to *Admin* rebec over the network. If *Admin* receives a report of dangerous gas levels, it notifies *Scientist* immediately and waits for *Scientist* acknowledge. If *Scientist* does not respond, *Admin* requests *Rescue* to reach and save *Scientist*. The main property to be checked is saving *Scientist* before the rescue deadline missed. We have varied the number of the sensors to produce state spaces of different sizes.

Slotted ALOHA Protocol The Slotted ALOHA protocol [2] is a network random access protocol which controls the data link medium access. Slotted ALOHA divide the time in to a number of slots and each network interface sends its data at the beginning of a time slot. We have modeled the Slotted ALOHA using four different reactive classes *User*, *Interface*, *Medium*, and *Controller*. *Controller* rebec is a police of medium (*Medium* rebec). *Interface*s of the model are waiting for *Contoller* signal and send the data via *Medium* afterwards. To make the model more realistic, we

Problem	Size	Using FTTS		Using Timed Automata	
		#states	time	#states	time
Ticket Service	**1 customer**	8	<1(sec)	801	<1(sec)
	2 customers	56	<1(sec)	19263811	≈ 5(hours)
	3 customers	20617	3(secs)	-	-
Sensor Net.	**1 sensor**	52	<1(sec)	-	-
	2 sensors	592	<1(sec)	-	-
	3 sensors	8154	1	-	-
	4 sensors	122900	18	-	-
Slotted ALOHA Protocol	**1 interface**	159	<1(sec)	-	-
	2 interfaces	2030	1(sec)	-	-
	3 interfaces	17253	5(secs)	-	-
	4 interfaces	132200	52(secs)	-	-
	5 interfaces	966147	490(secs)	-	-

Table 1. The model-checking time and state space size, using two different approaches

assigned a *User* to each *Interface* which provides *Interface* with requested data. We generated different size of models by varying the number of *User*s and *Interface*s.

Ticket Service. The detailed description of the *TicketService* has been explained at Section 2.1. Hitting the deadline of issuing the ticket is the desired property of this model. Varying the number of customers helped us to generate models of different sizes.

The Rebeca code of each case study can be found at the Rebeca homepage [1]. Table 1 shows the results of model checking these examples using the two different approaches. The model-checking time is limited to one day for each model. There is "-" as the results of model-checking for the models which take more than a day to model-check.

7. Conclusion

In this paper we introduced the floating-time transition system for schedulability and deadlock freedom analysis of Timed-Rebeca models. Floating-Time transition system exploits the key features of Timed-Rebeca. In summary, having no shared variables, no blocking send or receive, single-threaded actors, and non-preemptive execution of each message server give us an isolated message server execution, meaning that execution of a message server of a rebec will not interfere with execution of a message server of another rebec. Moreover, for checking schedulability and deadlock freedom we can focus only on events. In FTTS each transition shows releasing an event, or in other words execution of a message server of a rebec. Hence, in each state in FTTS rebecs may have different local times, but the transitions still gives us a correct order of release times of events of a specific rebec. We have proved a bisimulation relation between the SOS-based transition system and the floating-time transition system of Timed-Rebeca models. Our proposed approach is implemented as a part of Afra toolset [1]. Experimental evidence supports that direct model-checking of Timed-Rebeca models using floating-time transition system decreases both model-checking state space size and time consumption in comparison with translating to secondary models such as timed automata. Therefore, we can efficiently model-check more complex models.

In addition, our technique is based on the actor model of computation where the interaction is solely based on asynchronous message passing between the components. So, the proposed transition system and analysis techniques are general enough to be applied to similar computation models where they have message-driven communication and autonomous objects as units of concurrency such as agent-based systems.

References

[1] *Rebeca Home Page*. http://www.rebeca-lang.org.

[2] Norman Abramson. THE ALOHA SYSTEM: another alternative for computer communications. In *AFIPS '70 (Fall): Proceedings of the November 17-19, 1970, fall joint computer conference*, pages 281–285, New York, NY, USA, 1970. ACM.

[3] Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, Arni Hermann Reynisson, Steinar Hugi Sigurdarson, and Marjan Sirjani. Modelling and simulation of asynchronous real-time systems using timed rebeca. In Mohammad Reza Mousavi and António Ravara, editors, *FOCLASA*, volume 58 of *EPTCS*, pages 1–19, 2011.

[4] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA, 1990.

[5] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[6] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[7] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal - a tool suite for automatic verification of real-time systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hy-*

brid Systems, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer, 1995.

[8] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transaction on Programming Languages and Systems*, 8(2):244–263, 1986.

[9] Frank S. de Boer, Tom Chothia, and Mohammad Mahdi Jaghoori. Modular schedulability analysis of concurrent objects in creol. In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 5961 of *Lecture Notes in Computer Science*, pages 212–227. Springer, 2009.

[10] C. Hewitt. Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot. MIT Artificial Intelligence Technical Report 258, Department of Computer Science, MIT, April 1972.

[11] Carl Hewitt. What is commitment? physical, organizational, and social (revised). In *Proceedings of Coordination, Organizations, Institutions, and Norms in Agent Systems II*, Lecture Notes in Computer Science, pages 293–307. Springer, 2007.

[12] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular actor formalism for artificial intelligence. In Nils J. Nilsson, editor, *IJCAI*, pages 235–245. William Kaufmann, 1973.

[13] Mohammad-Javad Izadi. An actor based model for modeling and verification of real-time systems. Master's thesis, University of Tehran, School of Electrical and Computer Engineering, Iran, 2010.

[14] Mohammad Mahdi Jaghoori, Ali Movaghar, and Marjan Sirjani. Modere: The model-checking engine of Rebeca. In Hisham Haddad, editor, *SAC*, pages 1810–1815. ACM, 2006.

[15] Mohammad Mahdi Jaghoori, Marjan Sirjani, Mohammad Reza Mousavi, Ehsan Khamespanah, and Ali Movaghar. Symmetry and partial order reduction techniques in model checking Rebeca. *Acta Informatica*, 47(1):33–66, 2010.

[16] M. M. Jaghouri. *Time At Your Service: Schedulability Analysis Of Real-Time And Distributed Services*. PhD thesis, LIACS, December 2010.

[17] Dilsun Kirli Kaynar, Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Timed i/o automata: A mathematical framework for modeling and analyzing real-time systems. In *RTSS*, pages 166–177. IEEE Computer Society, 2003.

[18] Dilsun Kirli Kaynar, Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. *The Theory of Timed I/O Automata, Second Edition*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2010.

[19] Peter Csaba Ölveczky and José Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theor. Comput. Sci.*, 285(2):359–405, 2002.

[20] Shangping Ren and Gul Agha. Rtsynchronizer: Language support for real-time specifications in distributed systems. In Richard Gerber and Thomas J. Marlowe, editors, *Workshop on Languages, Compilers, & Tools for Real-Time Systems*, pages 50–59. ACM, 1995.

[21] Hamideh Sabouri and Marjan Sirjani. Slicing-based reductions for Rebeca. In *Proceedings of FACS 2008*. ENTCS, 2008.

[22] Marjan Sirjani, Frank S. de Boer, and Ali Movaghar-Rahimabadi. Modular verification of a component-based actor language. *J. UCS*, 11(10):1695–1717, 2005.

[23] Marjan Sirjani, Ali Movaghar, Amin Shali, and Frank S. de Boer. Modeling and verification of reactive systems using Rebeca. *Fundam. Inform.*, 63(4):385–410, 2004.

[24] Marjan Sirjani, Amin Shali, Mohammad Mahdi Jaghoori, Hamed Iravanchi, and Ali Movaghar. A front-end tool for automated abstraction and modular verification of actor-based models. In *ACSD*, pages 145–150. IEEE Computer Society, 2004.