

Timed Actors and Their Formal Verification

Marjan Sirjani

Mälardalen University
Västerås, Sweden

marjan.sirjani@mdu.se

Ehsan Khamespanah

University of Tehran
Tehran, Iran

e.khamespanah@ut.ac.ir

In this paper we review the actor-based language, Timed Rebeca, with a focus on its formal semantics and formal verification techniques. Timed Rebeca can be used to model systems consisting of encapsulated components which communicate by asynchronous message passing. Messages are put in the message buffer of the receiver actor and can be seen as events. Components react to these messages/events and execute the corresponding message/event handler. Real-time features, like computation delay, network delay and periodic behavior, can be modeled in the language. We explain how both Floating-Time Transition System (FTTS) and common Timed Transition System (TTS) can be used as the semantics of such models and the basis for model checking. We use FTTS when we are interested in event-based properties, and it helps in state space reduction. For checking the properties based on the value of variables at certain point in time, we use the TTS semantics. The model checking toolset supports schedulability analysis, deadlock and queue-overflow check, and assertion based verification of Timed Rebeca models. TCTL model checking based on TTS is also possible but is not integrated in the tool.

1 Introduction

Actors are introduced for modeling and implementation of distributed systems [7, 3]. Timed Actors allow us to introduce timing constraints, and progress of time, and are most useful for modeling time-sensitive systems. Timed Rebeca is one of the first timed actor languages with model checking support [10]. Timed Rebeca restricts the modeller to a pure asynchronous actor-based paradigm, where the structure of the model can represent the service oriented architecture, while the computational model matches the network infrastructure [1]. In a different context, it may represent components of cyber-physical systems, where components are triggered by events put in their input buffers, or by time events [14]. Timed Rebeca is equipped with analysis techniques based on the standard semantics of timed systems, and also an innovative event-based semantics that is tailored for timed actor models [13].

Timed Rebeca is an extension of the Reactive Object Language, Rebeca [16]. It is reviewed and compared to a few other actor languages in a survey published in ACM Computing Surveys in 2017 [4]. The very first ideas of Rebeca and its compositional verification is presented at AVoCS workshop in 2001 [15]. Timed Rebeca, different formal semantics of it, and the model checking support are presented in multiple papers. Here we present an overall view and insight into different semantics and use a simple example to show the differences visually.

2 Timed Rebeca

A Timed Rebeca model mainly consists of a number of *reactive class* definitions. These reactive classes define the behavior of the classes of the actors in the model. The model also has a `main` block that defines the instances of the actor classes.

We use a simple Timed Rebeca model as an example to explain the language features. In this example we consider two different actors. The first actor is able to handle three different tasks, named as *job1*, *job2*, and *job3*. The second actor can only handle one task, named as *job4*. The Timed Rebeca model of this example is shown in Listing 1. there are two classes of actors: Actor1 (lines 1-15) and Actor2 (lines 17-27). The main block in lines 29-32 defines one instance of each class. Each reactive class has a number of *state variables*, representing the local state of the actors. They may contain variables of basic data types, including booleans, integers, arrays, or references to other actors. To make the example model simple, none of the reactive classes of Listing 1 has any state variables. Each class can have a *constructor*, which is used to initialize the created instances of the class by initializing the state variables, and start up running of the model by sending messages to itself or other actors.

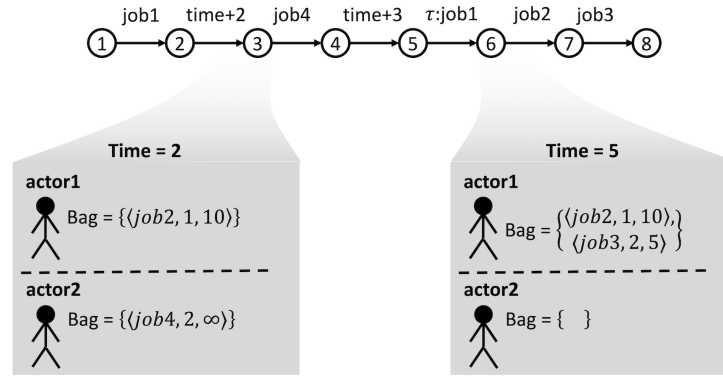
In the Timed Rebeca model of Listing 1, in the constructor of Actor1 (line 3), the actor sends itself a *job1* message. Each reactive class accepts a number of message types which are handled using *message servers* (*msgsrv*). Actor1 has three message servers, *job1* (lines 5-8), *job2* (lines 9-11), and *job3* (lines 12-14). Serving a message of type *job1* results in sending *job2* message to self which is put in the message buffer of itself only after passing 1 unit of time (modeled by using the *after* construct). The *deadline* construct denotes the deadline of the message to be handled, if at the time of handling the event the deadline is passed the model checking tool notifies that. Then, there is a *delay* statement which models progress of time for 5 units of time, this can be used to model a computation delay. In the definition of the message servers, well-known program control structures can be used, including *if-else* conditional statements, *for* and *while* loops, the definition of local variables, and assignments using usual arithmetic, logic, and comparative operators.

```

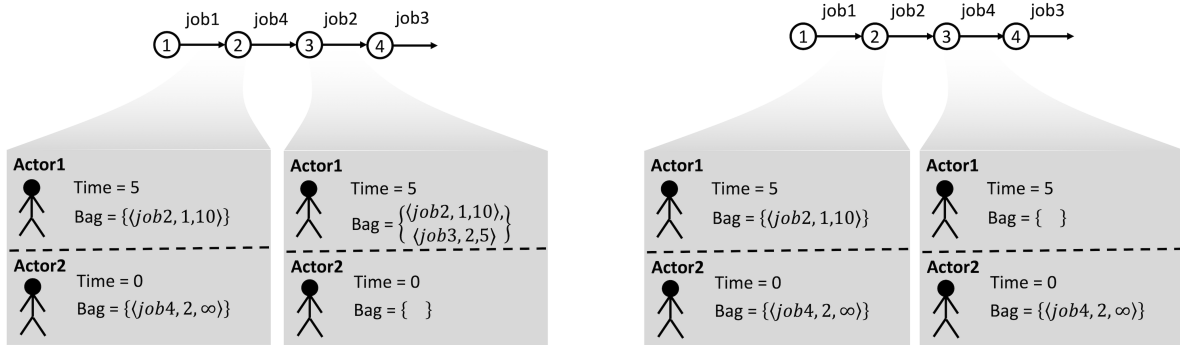
Listing 1: A simple Timed Rebeca model with two actors.
1  reactiveclass Actor1(3) {
2    Actor1() {
3      self.job1();
4    }
5    msgsrv job1() {
6      self.job2() after(1) deadline(10);
7      delay(5);
8    }
9    msgsrv job2() {
10   }
11   }
12   msgsrv job3() {
13     self.job3() after(1);
14   }
15 }
16
17 reactiveclass Actor2(3) {
18   knownrebecs {
19     Actor1 a1;
20   }
21   Actor2() {
22     self.job4() after(2);
23   }
24   msgsrv job4() {
25     a1.job3() after(2) deadline(5);
26   }
27 }
28
29 main {
30   Actor1 actor1():();
31   Actor2 actor2(actor1):();
32 }

```

In Timed Rebeca models, we assume that actors have local clocks which are synchronized throughout the model. Each message is tagged with a time stamp (called a time tag). We use a *delay(t)* statement to model the computation delay, and we use *after(t)* in combination with a send message statement to model a network delay, or model a periodic event. When we use *after(t)* in a send message statement it means that the time tag of the message when it is put in the queue of the receiver is the value of the local clock of the sender plus the value of *t*. The progress of time is forced by the *delay* statement.



(a) TTS of the Timed Rebeca model of Listing 1



(b) FTTS of the Timed Rebeca model of Listing 1. Each actor has its own local clock represented by Time. The value of local clock is considered in choosing the next transition.

(c) Relaxed-FTTS of the Timed Rebeca model of Listing 1. Each actor has its own local clock represented by Time. The message with the lowest time tag is chosen and the execution of its message server is the label of the next transition.

Figure 1: Comparing TTS, FTTS, and the relaxed form of FTTS for the Timed Rebeca model of Listing 1.

We assume that the local clock of each actor is zero when the model starts execution, the local clock is increased by value of τ if there is a `delay(τ)` statement. A `send` statement with an `after` does not cause an increase in the local time necessarily. The local time of the receiver actor is set to the time tag of the message when the actor picks the message, unless it is already greater than that. The latter situation means that the message sits in the queue while the actor is busy executing another message, in this case the `after` construct does not cause progress of time. The progress of time happens in the case that the time tag of the message is greater than the local time of the receiver actor, in this case the local time will be pushed forward. In Timed Rebeca, messages are executed atomically and are not preempted.

3 Different Semantics of Timed Rebeca

We first introduced an event-based semantics for Timed Rebeca and used McErlang for simulation of Timed Rebeca models in [1, 12]. In this semantics we focused on the object-based features of actors, encapsulation and information hiding, and decided on a coarse-grained semantics where serving a message (or handing a request or signal) are the only observable behavior of actors. We considered taking a message from top of the message queue and executing it as an observable action, and we called it an

event. Note that by a message queue in Timed Rebeca, we mean a bag of messages where each message has a time tag of when the message is put in the buffer. Here, by “top of the message queue of an actor”, we mean the message with the least time tag in the bag of messages targeted to that actor. In defining the formal semantics of Timed Rebeca as a labeled transition system, we only have one type of label on the transitions, *events*, which are taking messages and executing them. In [11], and its extended version [10], we introduced this event-based semantics of Timed Rebeca as Floating Time Transition System (FTTS) and compared it with the time semantics that is generally used for timed models (for example for Timed CCS [2]) where the transitions can be of the type of an event, progress of time, and a silent action.

Although we consider FTTS as the original and most fit semantics for Timed Actors, it may also be seen as a reduction technique in model checking. FTTS can give a significant reduction in state space compared to the standard Timed Transition System. In [10] we proved that there is an action-based weak bisimulation relation between FTTS and TTS of Timed Rebeca. Note that the focus here is on the labels on the transitions not on the values of variables in the states.

The semantics presented in [12], is a relaxed form of FTTS in [10] where in choosing the next step in a state we have a simpler policy. The SOS rules of FTTS and the relaxed version are presented in [10] and [12], respectively. In each state, the SOS rule for the scheduler chooses the next message in the bags of actors to be executed. In the relaxed form of FTTS, the scheduler simply chooses the message with the least time tag (targeted to any actor). In FTTS, the scheduler considers the local clock of each actor as well. For each actor, the maximum between the local clock and the lowest time tag of the messages in the message bag of the actor is computed. Then among all the actors, the scheduler chooses the actor with the least of these amounts. The message on the top of the queue of this chosen actor will be executed next. Comparing to the standard TTS, the relaxed form of FTTS preserves the order of execution of messages of all actors if we consider the time tags of messages for ordering. The intuitive reason is that in Timed Rebeca we consider a FIFO policy for scheduling the messages in the message buffer, when we choose the message with the lowest time tag to be executed, it is guaranteed that from that point on, there will be no messages with a smaller time tag added to the message buffer (of any actor). So, the FIFO policy for serving messages can be correctly respected. The subtle point here is that the actor a with the lowest time tag message m may be busy when message m is sitting in its message buffer, in the meanwhile other messages from other actors may get the chance to be executed and send messages to actor a . Of course, the time tag of those messages will be greater than the time tag of message m , but still we are losing the “correct” content of the message buffer of a at some snapshots in time. By this observation, we moved to the FTTS semantics in [10] where at any point in time, we have the correct content of the message buffer. Using this semantics we may choose to use other scheduling policies for messages (events) in the buffer, for example the *earliest deadline first* policy.

In Figure 1, we show parts of the the state transition system for Rebeca model in Listing 1. In this figure, we see how in TTS we may have three types of labels on the transitions, an event, time progress and τ (silent) transitions. In FTTS, in state 2 in Figure 1.b, the scheduler chooses the message $\langle \text{job4}, 2, \infty \rangle$ while in the relaxed form of FTTS, in Figure 1.c, the scheduler chooses the message $\langle \text{job2}, 1, 10 \rangle$. The reason is that although the message with the lowest time tag is $\langle \text{job2}, 1, 10 \rangle$, with the time tag 1, the maximum between 1 and the value for the local clock of Actor1 is 5. The maximum between 2 (the time tag for message $\langle \text{job4}, 2, \infty \rangle$) and the value for the local clock of Actor2 (which is zero in this state) is 2.

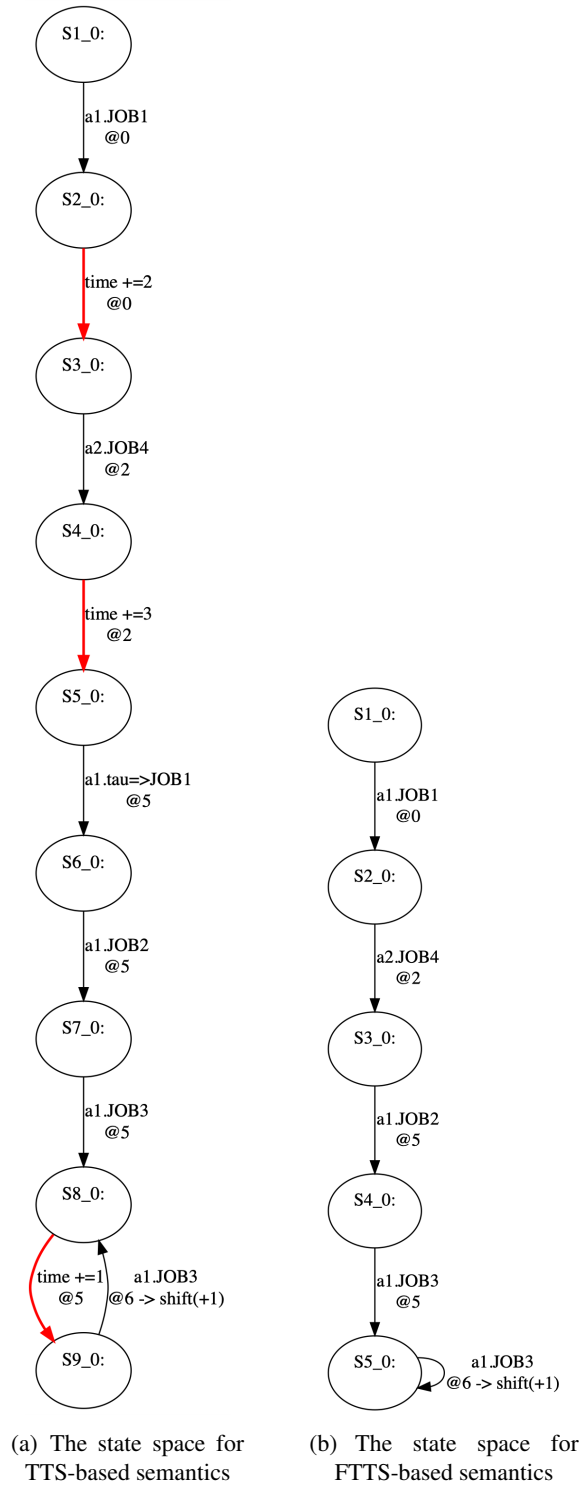


Figure 2: The state space of the Rebeca model of Listing 1, generated by Afra using TTS and FTTS semantics

4 Model Checking Timed Rebeca Models

The verification algorithms of TTS with dense time are generally PSPACE-complete as stated in [6]. In the existing model checking tools commonly the properties are limited to a subset of TCTL properties without nested timed quantifiers. For this subset efficient algorithms are developed. In the case of Timed Rebeca, we use *discrete time*, and hence TTS can be verified efficiently in polynomial time against TCTL properties. Discrete time is the time model in which passage of time is modeled by natural numbers. We developed a model checking tool and a reduction technique for Timed Rebeca models based on TTS semantics against TCTL properties [8]. This toolset is not integrated in the Afra IDE [9].

We also developed a tool for the model checking of Timed Rebeca models based on both TTS and FTTS semantics, which is integrated in Afra. The current implementation of the model checking toolset supports schedulability analysis, and checking for deadlock-freedom, queue-overflow freedom, and assertion based verification of Timed Rebeca models. Note that in FTTS, in each state actors may have different local clocks, so, writing meaningful assertion needs special care. Assertions on state variables of one actor are not problematic. The Timed Rebeca code of the case studies and the model checking toolset are accessible from Rebeca homepage [5].

Figure 2 shows the state space generated automatically by the model checking tool, Afra, for the Timed Rebeca model in Listing 1 based on the two semantics, TTS and FTTS. It is shown that the order of events are preserved while time progress and τ transitions are hidden. In state *S9_0* in Figure 2.a, and state *S5_0* in Figure 2.b, you see how the transition system becomes bounded using a shift operation on the time. The shift keyword means that for example by the event *a1.J0B3*, we go back to state *S8_0* (or *S5_0*), where all the values of state variables, local variables and messages in the message buffers stay the same but the value of parameters related to time (including time tag of all messages and local clock value) change and have a shift by the same value.

References

- [1] Luca Aceto, Matteo Cimini, Anna Ingólfssdóttir, Arni Hermann Reynisson, Steinar Hugi Sigurdarson & Marjan Sirjani (2011): *Modelling and Simulation of Asynchronous Real-Time Systems using Timed Rebeca*. In Mohammad Reza Mousavi & António Ravara, editors: *Proceedings 10th International Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA 2011, Aachen, Germany, 10th September, 2011, EPTCS 58*, pp. 1–19, doi:10.4204/EPTCS.58.1.
- [2] Luca Aceto, Anna Ingólfssdóttir, Kim Guldstrand Larsen & Jiri Srba (2007): *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, USA, doi:10.1017/CBO9780511814105.
- [3] Gul Agha (1986): *Actors: a model of concurrent computation in distributed systems*. MIT press, doi:10.7551/mitpress/1086.001.0001.
- [4] Frank S. de Boer, Vlad Serbanescu, Reiner Hähnle, Ludovic Henrio, Justine Rochas, Crystal Chang Din, Einar Broch Johnsen, Marjan Sirjani, Ehsan Khamespanah, Kiko Fernandez-Reyes & Albert Mingkun Yang (2017): *A Survey of Active Object Languages*. *ACM Comput. Surv.* 50(5), pp. 76:1–76:39, doi:10.1145/3122848.
- [5] Rebeca Research Group: *Afra toolset homepage*. Available at <http://rebeca-lang.org/alltools/Afra>.
- [6] Thomas A. Henzinger, Zohar Manna & Amir Pnueli (1991): *Timed Transition Systems*. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever & Grzegorz Rozenberg, editors: *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings, Lecture Notes in Computer Science 600*, Springer, pp. 226–251, doi:10.1007/BFb0031995.

- [7] Carl Hewitt (1977): *Viewing control structures as patterns of passing messages*. *Artificial intelligence* 8(3), pp. 323–364, doi:10.1016/0004-3702(77)90033-9.
- [8] Ehsan Khamespanah, Ramtin Khosravi & Marjan Sirjani (2018): *An efficient TCTL model checking algorithm and a reduction technique for verification of timed actor models*. *Sci. Comput. Program.* 153, pp. 1–29, doi:10.1016/j.scico.2017.11.004.
- [9] Ehsan Khamespanah, Ramtin Khosravi & Marjan Sirjani (2023): *Afra: An Eclipse-Based Tool with Extensible Architecture for Modeling and Model Checking of Rebeca Family Models*. In Hossein Hojjat & Mieke Massink, editors: *Fundamentals of Software Engineering - 11th International Conference, FSEN 2023, May 4-5, 2023*, Lecture Notes in Computer Science, Springer, doi:10.1007/978-3-031-42441-0_6.
- [10] Ehsan Khamespanah, Marjan Sirjani, Zeynab Sabahi-Kaviani, Ramtin Khosravi & Mohammad-Javad Izadi (2015): *Timed Rebeca schedulability and deadlock freedom analysis using bounded floating time transition system*. *Sci. Comput. Program.* 98, pp. 184–204, doi:10.1016/j.scico.2014.07.005.
- [11] Ehsan Khamespanah, Marjan Sirjani, Mahesh Viswanathan & Ramtin Khosravi (2015): *Floating Time Transition System: More Efficient Analysis of Timed Actors*. In Christiano Braga & Peter Csaba Ölveczky, editors: *Formal Aspects of Component Software - 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers*, Lecture Notes in Computer Science 9539, Springer, pp. 237–255, doi:10.1007/978-3-319-28934-2_13.
- [12] Arni Hermann Reynisson, Marjan Sirjani, Luca Aceto, Matteo Cimini, Ali Jafari, Anna Ingólfssdóttir & Steinar Hugi Sigurdarson (2014): *Modelling and simulation of asynchronous real-time systems using Timed Rebeca*. *Sci. Comput. Program.* 89, pp. 41–68, doi:10.1016/j.scico.2014.01.008.
- [13] Marjan Sirjani & Ehsan Khamespanah (2016): *On Time Actors*. In Erika Ábrahám, Marcello M. Bonsangue & Einar Broch Johnsen, editors: *Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science 9660, Springer, pp. 373–392, doi:10.1007/978-3-319-30734-3_25.
- [14] Marjan Sirjani, Edward A. Lee & Ehsan Khamespanah (2020): *Verification of Cyberphysical Systems*. *Mathematics* 8(7), doi:10.3390/math8071068.
- [15] Marjan Sirjani, Ali Movaghar & Mohammadreza Mousavi (2001): *Compositional verification of an object-based reactive system*. In: *Workshop on Automated Verification of Critical Systems AVoCS 2001*.
- [16] Marjan Sirjani, Ali Movaghar, Amin Shali & Frank S. de Boer (2004): *Modeling and Verification of Reactive Systems using Rebeca*. *Fundam. Informaticae* 63(4), pp. 385–410. Available at <http://content.iospress.com/articles/fundamenta-informaticae/fi63-4-05>.