UPPAAL
Software Fiável 2019–2020

Pedro Carrega 49480
Vasco Ferreira 49470

2019/12/20

Abstract

This essay describes the tool UPPAAL, a tool that can be used for modeling and verification of real-times systems while being more user friendly, and in some regards, more powerful than other tools like SPIN.

1 Introduction

UPPAAL is a tool for modeling, validation and verification of real-time systems represented by (a network of) timed automata extended with integer variables, structured data types, and channel synchronization. It was developed in collaboration between the Uppsala University located in Sweden and the Aalborg University in Denmark. The main problem was that timed automats weren't powerful and expressive enough so it could capture behaviors of complex cyber-physical systems. With that in mind it was created a new branch of UPPAAL called UPPAAL SMC.

UPPAAL SMC proposes to represent systems via networks of automata whose behaviors may depend on both stochastic and non-linear dynamical features. Concretely, in UPPAAL SMC, each component of the system is described with an automaton whose clocks can evolve with various rates. Such rates can be specified with, e.g., ordinary differential equations.

The core idea of SMC is to monitor some simulations of the system, and then use results from the statistics area (including sequential hypothesis testing or Monte Carlo simulation) in order to decide whether the system satisfies the property with some degree of confidence. By nature, SMC is a compromise between testing and classical model checking techniques. Simulation-based methods are known to be far less memory and time intensive than exhaustive ones, more expressive and are oftentimes the only option. The tool is not suited for complex systems.

2 Installation

The tool can be acquired from their website. After filling a short form it will automatically download a zip with a folder inside. Extract that folder to anywhere you like and run the jar inside, opening the tool.

3 Usage

On opening of the tool you are presented with the following screen:

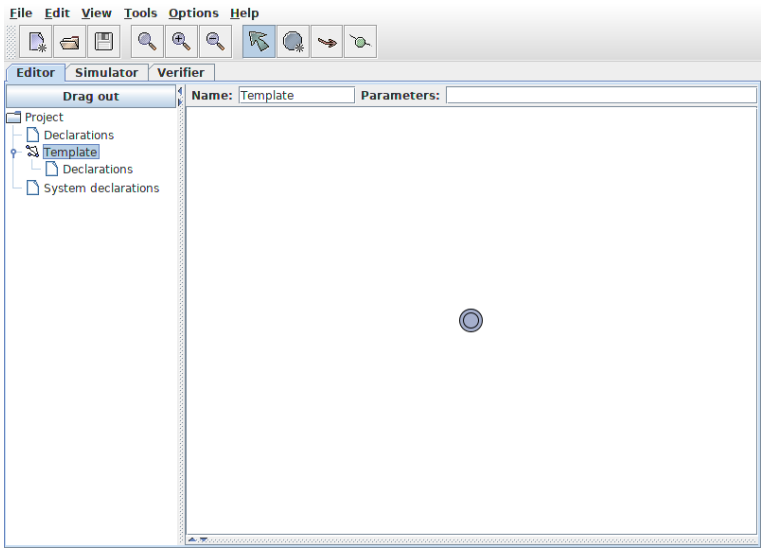


Figure 1: Startup View

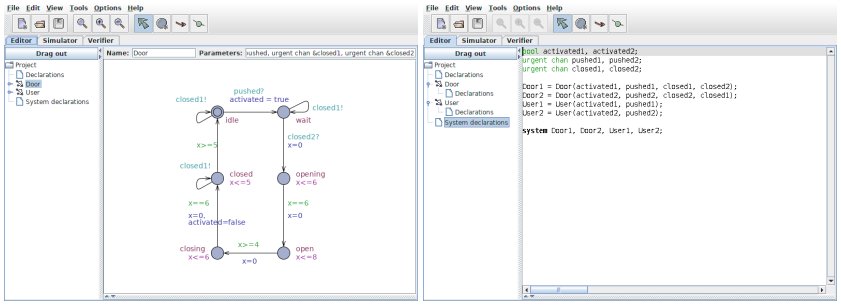
When starting a new project in UPPAAL you start with a system with an empty canvas and its declarations, the global declarations and the system declarations. In the editor screen you are able to model your system from the variables to the behavior of said system.

You start by declaring the different systems you will be using in your model. Systems can receive parameters, and such, they need to be declared in the system declarations. Said parameters need to have their type and name declared in the view of each system. In the declarations page you initiate the global variables of your system, where the local variables are initiated in the declarations page of each individual system. To model your system you are able to insert three different objects:

- Location
- Rail
- Nail

A location is a possible state of the system, a rail connects different locations and a nail is used on rails so you can model the graphical layout of the rail. In a location is possible to give it an invariant

In the end of modeling your system the editor page might look like the following figures.



(a) Model View (b) Variables View

In the simulator tab, like the name states, it's possible to simulate the working of the designed system.

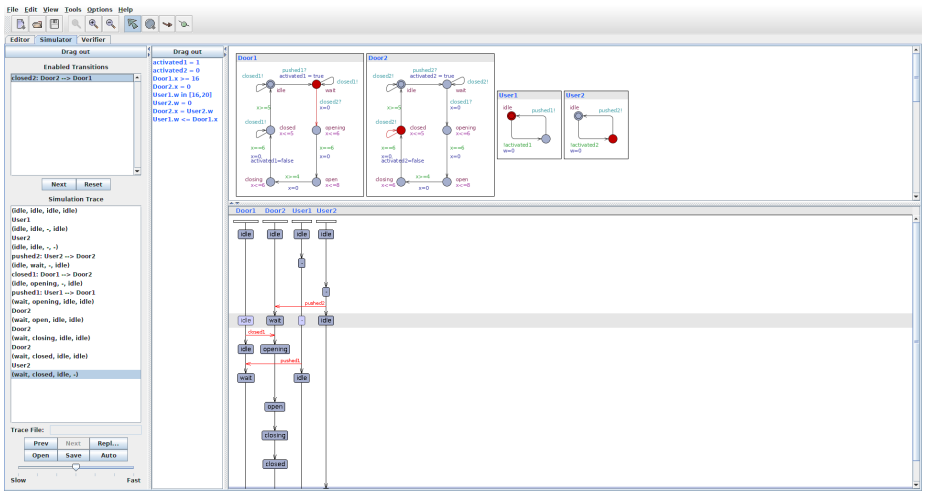


Figure 3: Simulator Tab

You are able to simulate step by step, the tool tracks the individual value of every variable in the system and every action taken by the simulation in order. The tool also offers a visual feedback of the execution tree of the system and the current state of each automata.

The last available tab is the Verifier tab. In this tab it is possible to the your system with various different properties.

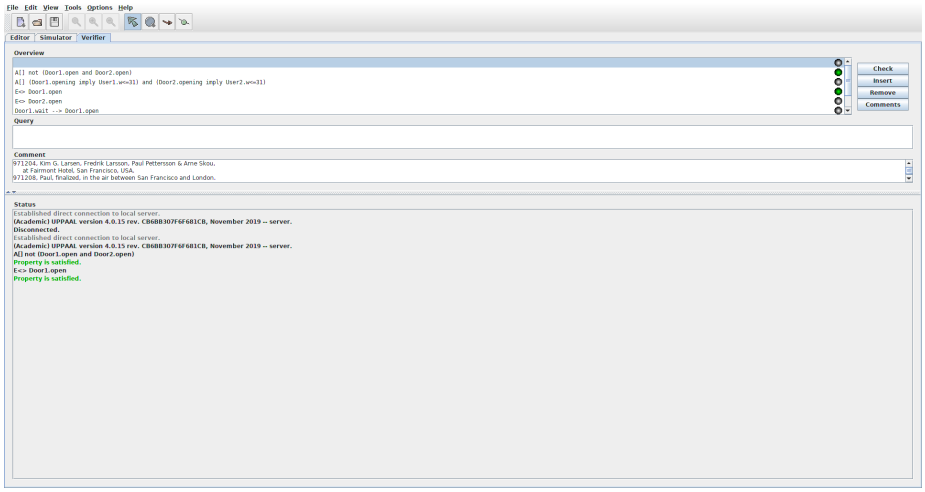


Figure 4: Verifier Tab

After writing a query the tool then tests the system, informing the user if the system meets the tested property or not.

4 Demo

In this section we will show you a possible modeling of the 2doors problem. The 2doors problem consists in the existence of two doors and two users, in which, only one door can be opened at a time. The users are attributed a single door and are responsible for triggering the opening of said door. For this system we are going to need two doors and two users, each door and user will have a boolean signaling if the door is open and each door will have three channels, one for communicating with the assigned user and two for communicating with the other door. The users will each have two channels, both to communicate with the assigned door.
So the system declarations would possibly look like the following figure:

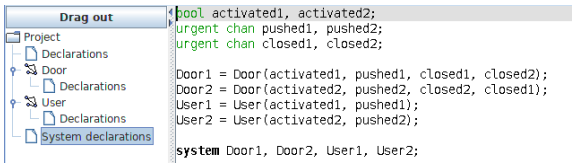


Figure 5: System Variables

Next we need to model our system. The user will only have two locations, one for when he is idle and other for when he pushes the button to open the door. From the idle state, the user can go only go push the button if said button is not activated. Having pushed the button the user uses the channel to inform the door that the button was pushed. The User model may look like the following figure:

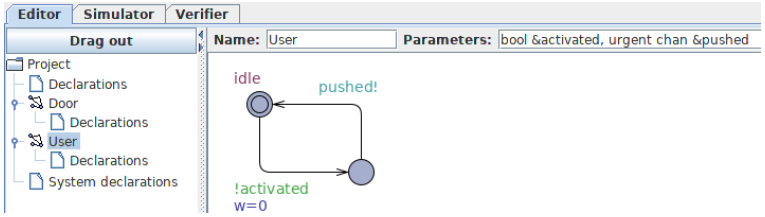


Figure 6: User Model

From the idle state the door will be always informing the other door that it is closed. will be listening, from the channel, if the button was pushed. When the button is pushed, the boolean indicating the status of the button is turned true. In the next location the door will keep broadcasting to the other door that is closed, while also listening to know if the other door is closed. When the guard is meet, the timer is set to 0. In the following location the door will start opening and when 6 seconds pass the guard of the rail is meet and the transition can be made. The timer is set again to 0 and the door is now opened. The door will then stay open for a minimum of 4 seconds and a maximum of 8, then the transition to the next location is made the timer is again set to 0. Now it will start closing and when exactly 6 seconds pass the door will be closed, setting the timer to 0 and resetting the activated button. It then starts broadcasting that the door is closed so that other doors are able to open. The mechanism will be locked and then returned to the idle state after 5 seconds pass. And modeling example for the Door model would be:

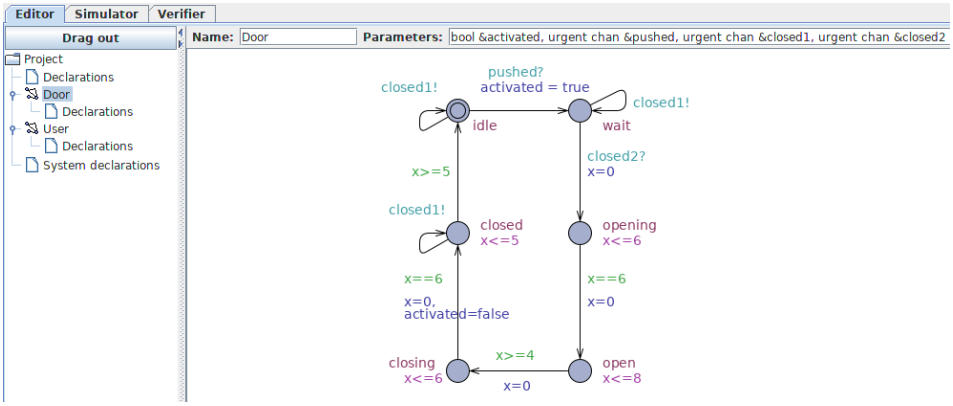


Figure 7: Door Model

In the following figure it's possible to verify the procedure of the system that was described in the implementation of the system.

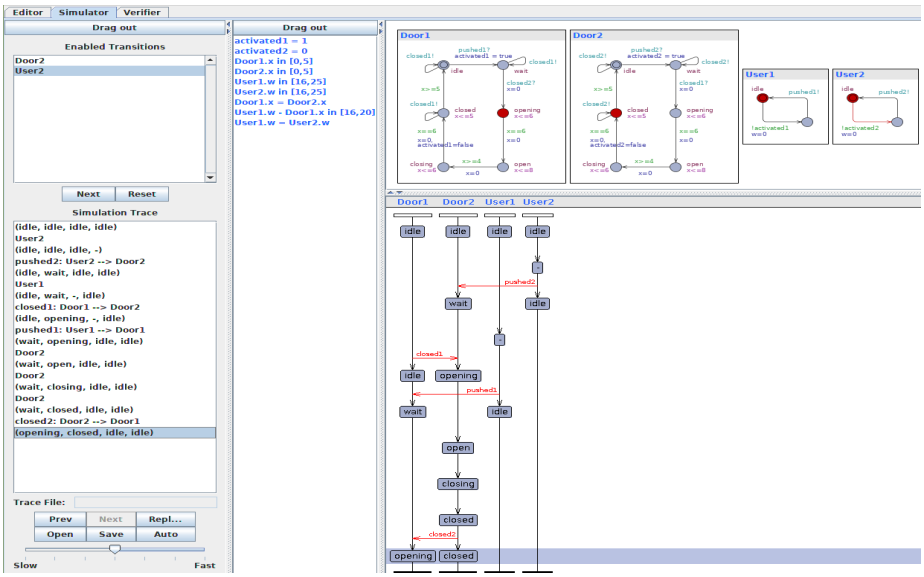


Figure 8: Simulation Tab

To finish the modeling of our system is necessary to check if all the properties mentioned in the problem are verified. The properties necessary to verify are:

- Only one door will be open at a time
- Each door will eventually open
- When a button is pushed the door will eventually open
- The system is deadlock-free

As you can see in the following figure, all the properties are satisfied:

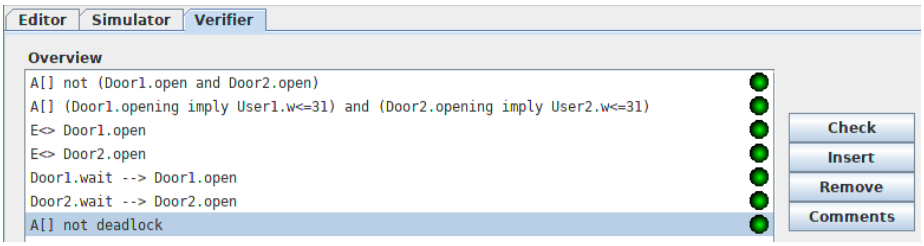


Figure 9: Verifier Tab

5 Strengths and Weaknesses

One of the obvious strengths of using a tool like UPPAAL is the possibility of modeling your system in real-time. It also offers the concept of "clocks", so you can define how much time a procedure should take. So that time is also a consideration of our system. Although SPIN also offers temporal logic, UPPAAL you can specify timers which allows increasing the complexity of the temporal logic of some properties of our system. UPPAAL also permits simulating the model step by step. The tool also offers a more intuitive UI when compared to SPIN's which lowers the learning curve of the tool.

When compared with SPIN, UPPAAL doesn't offer a counter-example which makes corrections more difficult. Regarding the verification of the system, UPPAAL only allows properties restricted to invariance and possibility properties. If a user wants to verify other properties as, for example, bounded liveness properties need to be expressed as a separate test.

6 Conclusion

UPPAAL is a model checking tool that presents itself as an alternative to SPIN having some advantages like the concept of clocks and the more user friendly interface. It offers a stochastic verification of the system with a high level of reliability. A user that isn't familiar with programming isn't handicapped by the tool since it relies on automats instead of specifying the behavior of the system through code. The tool is not perfect though, since it's limited in the type of verifications possible and don't offer any hints of solution when problems in the system arise.

References

[1] Uppaal.

[2] Alexandre David, Tobias Amnell, Martin Stigge, and Pontus Ekberg. Uppaal4.0 : Small tutorial, 2019.

[3] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. 2018.

[4] Henrik Ejersbo, JensenKim G., and LarsenArne Skou. Modelling and analysis of a collision avoidance protocol using spin and uppaal. *BRICS Report Series*, 1996.

[5] Vasco T. Vasconcelos and Antónia Lopes. Model checking, 2019.