Project no.:	**ICT-FP7-STREP-214755**
Project full title:	**Quantitative System Properties in Model-Driven Design**
Project Acronym:	**QUASIMODO**

Deliverable no.: D3.7

Title of Deliverable: Code Generation from Timed Models

Contractual Date of Delivery to the CEC:	**M40**
Actual Date of Delivery to the CEC:	**June 3, 2011**
Organisation name of lead contractor for this deliverable:	**AAU**
Author(s):	**Kim G. Larsen, Alexandre David Jean-Francois Raskin, Nicolas Markey**
Participants(s):	**AAU CFV CNRS**
Work package contributing to the deliverable:	**WP 1–4**
Nature:	**P**
Version:	**1.1**
Total number of pages:	**12**
Start date of project:	1 Jan. 2008 **Duration:** 36 month

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

Dissemination Level

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This deliverable describes the development of support for generation of executable code from timed specification. The contributions falls into two categories. Firstly, new methods for robustness of timed automata are given ensuring that implementation of the model on a platform with drift and/or imprecision will not introduce new behaviour. Secondly, a method, tool integration and demonstration has been given for systematics and automatic translation of winning strategies providing by UPPAAL-TIGA into simulation code for SIMULINK as well as executable code.

Keyword list: Robustness for timed automata. Controller systhesis, strategies, UPPAAL-TIGA, SIMULINK, executable code.

Contents

Abbreviations

AAU: Aalborg University, DK

CFV: Centre Fèdèrè en Vèrification, B

CNRS: National Center for Scientific Research, FR

ESI: Embedded Systems Institute, NL

ESI/RU: Radboud University Nijmegen, NL

RWTH: RWTH Aachen University, D

SU: Saarland University, D

1 Robustness: Transfer of Properties of Timed Automata Models to Implementations

Timed automata are governed by an idealized semantics that assumes a perfectly precise behavior of the clocks. The traditional semantics is not robust because the slightest perturbation in the timing of actions may lead to completely different behaviors of the automaton. During the first year of Quasimodo (deliverable D3.1) several works have considered a relaxation of this semantics, in which guards on transitions are widened by $\Delta > 0$ and clocks can drift by $\epsilon > 0$. The relaxed semantics encompasses the imprecisions that are inevitably present in an implementation of a timed automaton, due to the finite precision of digital clocks. Here we report on a number of new results.

We also point to the recent paper [4] for an introduction and survey of robustness results for timed automata.

1.1 Quantitative Robustness Analysis of Flat Timed Automata

1.1.1 Participants

- Remi Jaubert, Pierre-Alain Reynier, CNRS, Marseille; France.

1.1.2 Contribution

Whereas formal verification of timed systems has become a very active field of research, the idealized mathematical semantics of timed automata cannot be faithfully implemented. Recently, several works have studied a parametric semantics of timed automata related to implementability: if the specification is met for some positive value of the parameter, then there exists a correct implementation. In addition, the value of the parameter gives lower bounds on sufficient resources for the implementation. In [3], we present a symbolic algorithm for the computation of the parametric reachability set under this semantics for flat timed automata. As a consequence, we can compute the largest value of the parameter for a timed automaton to be safe.

1.2 Timed Automata Can Always Be Made Implementable

1.2.1 Participants

- Patricia Bouyer, Nicolas Markey, Ocan Sankur, LSV CNRS, Cachan; France.
- Kim G. Larsen, Claus Thrane, Aalborg University; Denmark.

1.2.2 Contribution

Timed automata follow a mathematical semantics, which assumes perfect precision and synchrony of clocks. Since this hypothesis does not hold in digital systems, properties proven formally on a timed automaton may be lost at implementation. In order to ensure implementability,

several approaches have been considered, corresponding to different hypotheses on the implementation platform.

A existing prominent approach, for verifying the behavior of real-time programs executed on CPUs, is robust model-checking. It consists in studying the enlarged semantics of the timed automaton, where all the con- straints are enlarged by a small (positive) perturbationΔ, in order to model the imprecisions of the clock. In some cases, this may allow new behaviours in the system, regardless of Δ. Such automata are said to be none-robust with respect to small perturbations. On the other hand, if no new behaviour is added to the system, that is, if the system is robust, then implementability on a fast-enough CPU will be ensured.

In [1], we show that timed automata can always be made implementable. More precisely it is shown, that from any timed automaton $\mathcal{A}$, we build a timed automaton $\mathcal{A}$ that exhibits the same behaviour as $\mathcal{A}$, and moreover $\mathcal{A}$ is both robust and samplable by construction.

2 Controller Synthesis

2.1 From Timed Games via Strategies to Executable Code

2.1.1 Participants

- Alexandre David, Jan Jakob Jessen, Kim G. Larsen, Jacob Illum, Aalborg University; Denmark.

2.1.2 Contribution

In the HYDAC case study we have successfully applied the tools UPPAAL-TIGA, PHAVER and SIMULINK to synthesize and implement a provably correct, robust and near-optimal controller. In [2] we have generalized this succesfull application into a systematic method for linking UPPAAL-TIGA and SIMULINK. In particular given a user defined timed game model in UPPAAL-TIGA, a winning strategy can be automatically imported to SIMULINK as an S-function for simulation, validation and automatic generation of executable code. For demonstration purposes, we have applied the methodology to a small two-tank example.

The framework requires that two models of the control problem are provided: an abstract model in terms of a timed game and a complete, dynamic model of the environment in terms of a hybrid system. Given the abstract (timed game) model together with logically formulated control and guiding objectives, UPPAAL-TIGA automatically synthesises a strategy which is directly compiled into an S-function[1]. This enables evaluation of the performance of the control strategy on the given environment by simulation in SIMULINK. Also, by choosing different control objectives for the synthesis problem in UPPAAL-TIGA, we can easily obtain and evaluated alternative controllers. Generation of final executable code is possible through the SIMULINK real-time workbench.

Tool Integration Figure 2 shows a more detailed view of the tool integration that we have implemented. The implementation is separated into one (internal) MATLAB function that acts as the coordinator component and a Ruby script that makes the translation from a strategy to an S-function. The user defines a timed game automaton model in UPPAAL-TIGA together with a SIMULINK model that contains a block in which the user wishes to insert the generated controller from UPPAAL-TIGA. It is up to the user to define the inputs and the outputs variables. These inputs and outputs are defined in SIMULINK and their names must match the corresponding variables in the UPPAAL-TIGA model. The user should make sure that the desired property is satisfied to obtain a strategy. Then the user calls the MATLAB function that

1. calls UPPAAL-TIGA to generate the strategy,
2. extracts the inputs and outputs from the SIMULINK model and generates input and output files,

[1] S-function is a term used in SIMULINK for executable content that can be embedded into its block components. S-functions support multiple languages such as C and MATLAB representation of the controller.

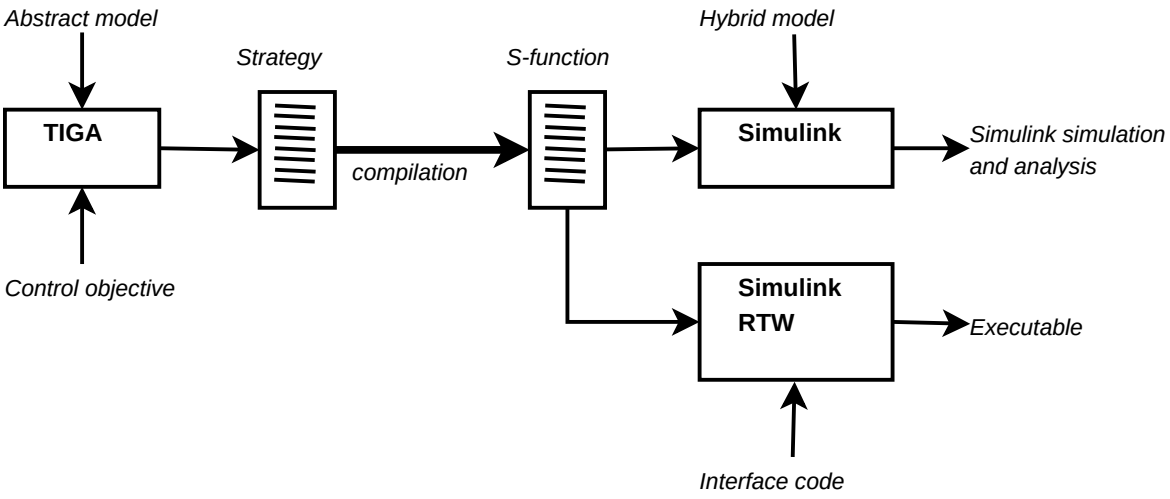


Figure 1: Overview of the framework.

3. calls the Ruby script that translates the strategy together with the declaration files of inputs and outputs into an S-function,

4. and calls the MATLAB C-compiler to compile the generated S-function and imports the binary into SIMULINK.

The SIMULINK model can now be simulated with the generated controller or it can be used with the real-time workbench to generate code from the S-function.

Demonstrator As a small demonstrator of the tool integration we have considered the 2-tank example of figure 3.(a). The idea is to maintain the temperature of two tanks containing some liquid within some specified bounds. We have one heater that can be used to heat either one of the two tanks but changing tank takes time. The temperature of the tanks should be kept between a safe middle range and in our abstraction we consider critical low or high temperatures that we do not want to reach and two ranges of temperatures that are observable by our controller. These serve as low and high thresholds as shown in the figure. The modeling of the dynamic evolution of the temperature is given a simple hybrid automaton (one for each tank) as shown in figure 3.(b). The two modes corresponds to the heater being on or off and with assocated differential equations to describe how the temperature changes. T is the temperature, K_1, K_2, and C are constants.

We model this system in UPPAAL-TIGA with one process per tank and one for the controller. Figure 4 shows the templates for the tank and the controller. The tank automaton (Fig. 4.(a)) reflects the two states of the heater being on and off and a clock $\mathtt{x}$ is used to measure time.

Temperature changes are then mapped to time intervals and the model is designed to take uncertainties into account. Figure 5 shows the principle. In Fig. 5.(a) the temperature decreases from somewhere from the high observable range to the lower one. We derive a lower an upper bound on time for detecting the state change. Similarly we derive time bounds when the temper-

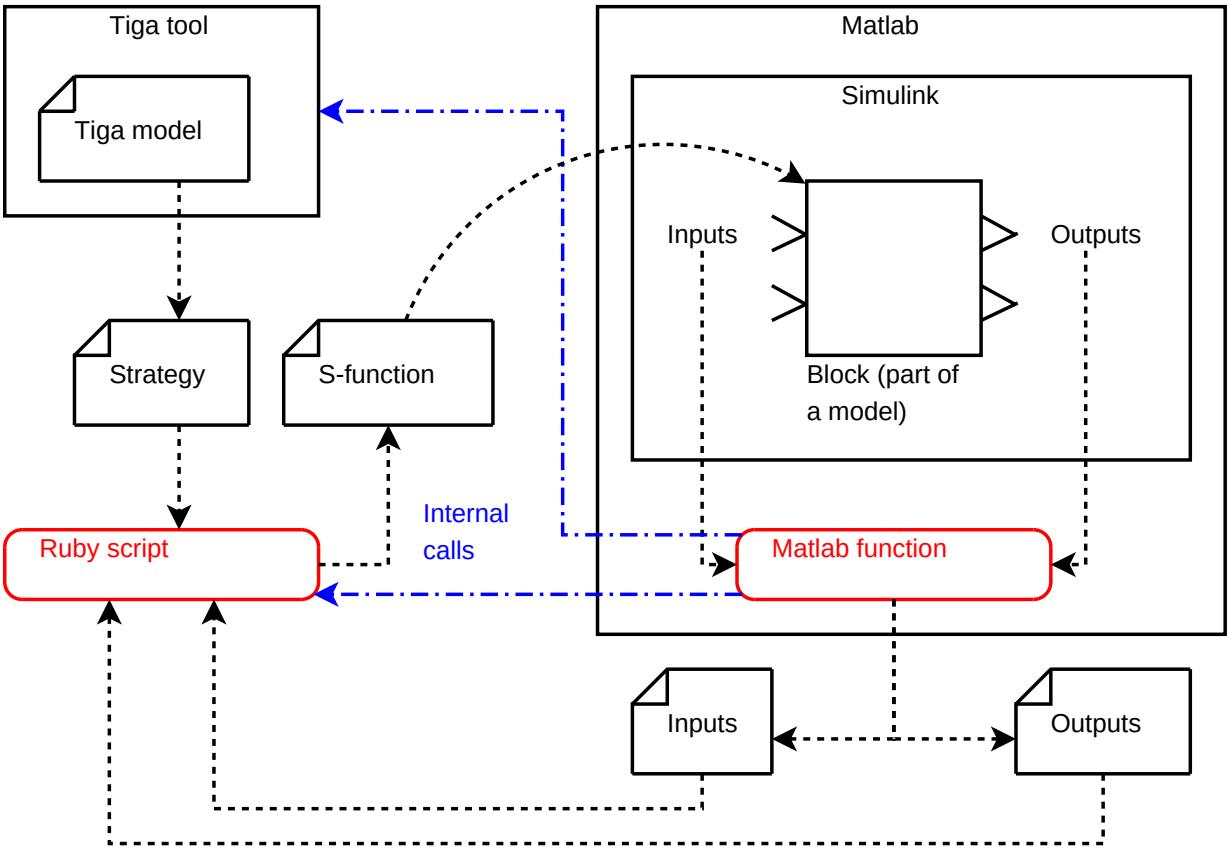


Figure 2: Integration of UPPAAL-TIGA and SIMULINK.

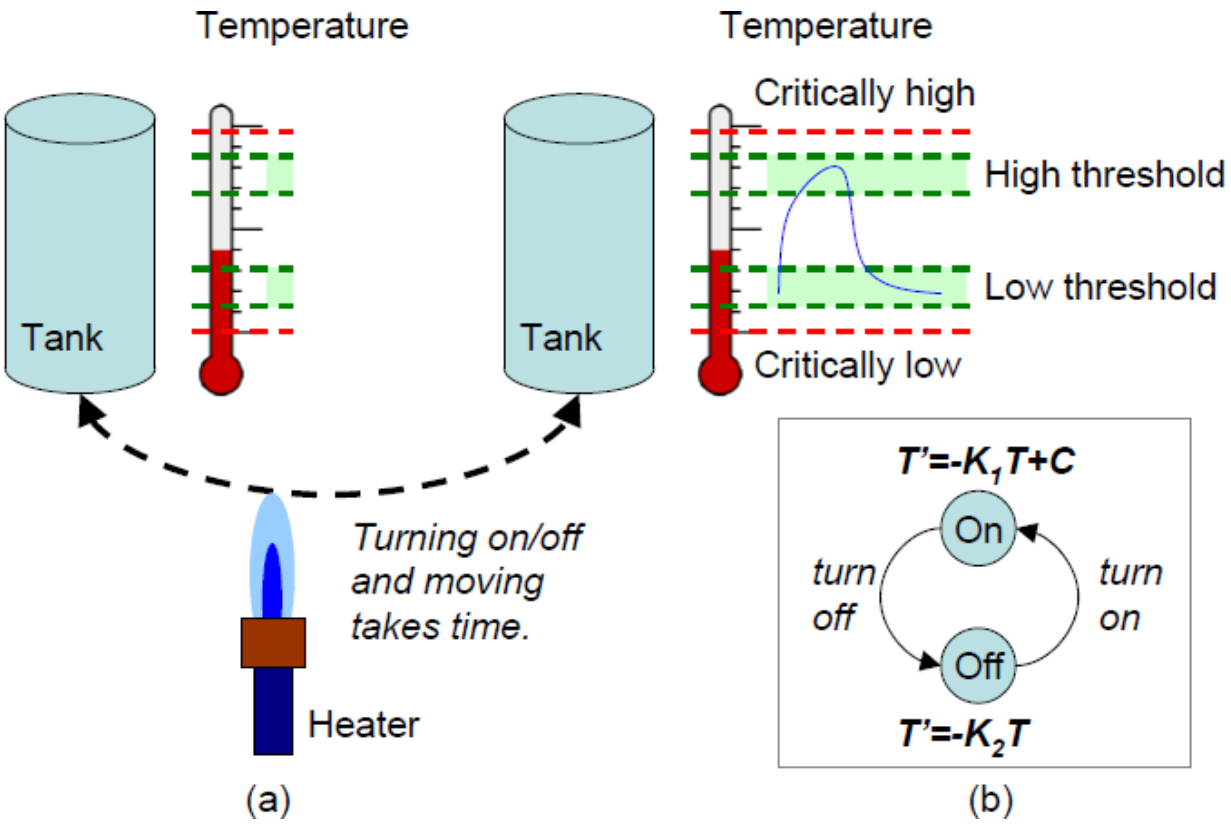


Figure 3: The 2-tank example. One heater can heat one tank at a time and moving the heater between the tanks takes time (a). The temperature of the tanks should stay within an acceptable range. The temperature is modelled by the simple hybrid system in (b) with two states associated with differential equations.

ature increases in Fig. 5.(b). The lower bounds are modelled by the guards (`GUARD_DEC[temp]` and `GUARD_INC[temp]` when the temperature is decreasing or increasing) and the upper bounds are the invariants (`TEMP_DEC[temp]` and `TEMP_INC[temp]` depending on heating). The model is designed to discretise an arbitrary number of such observable ranges and we make experiments with two and three such ranges. The controller (Fig. 4.(b)) models that it can turn a heater on or off with a constraint on time.

For the experiments below we consider the following temperature:

- Above 100, temperature is critical high (`temp=3`).
- Between 70 and 90, temperature is high (`temp=HIGH=2`).
- Between 40 and 60, temperature is low (`temp=LOW=1`).
- Below 30, temperature is critical low (`temp=0`).

The corresponding time intervals in the models are declared as follows:

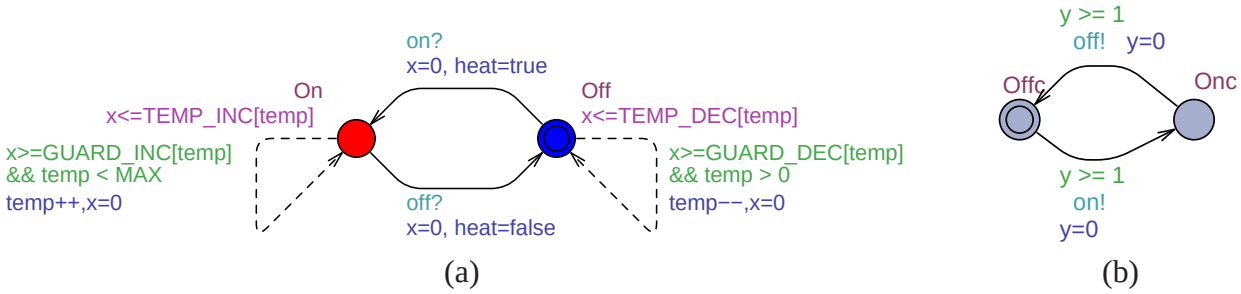


Figure 4: The model of the 2-tank example in UPPAAL-TIGA.

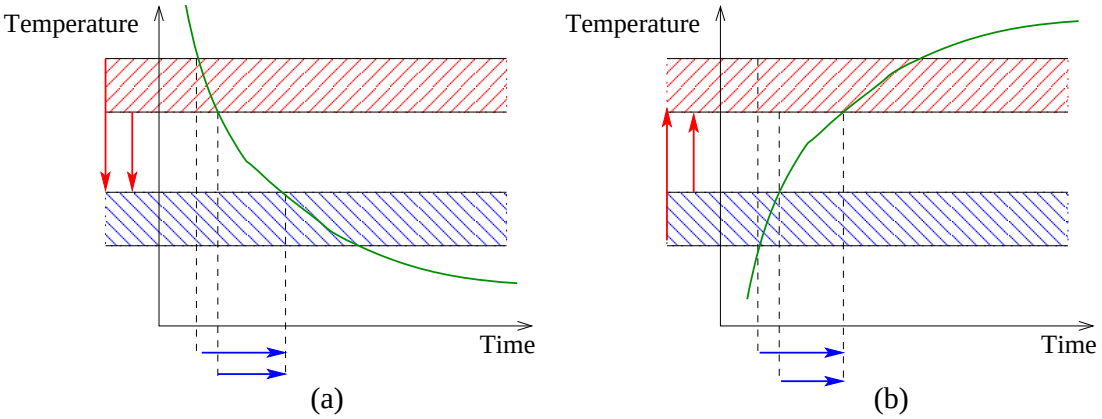
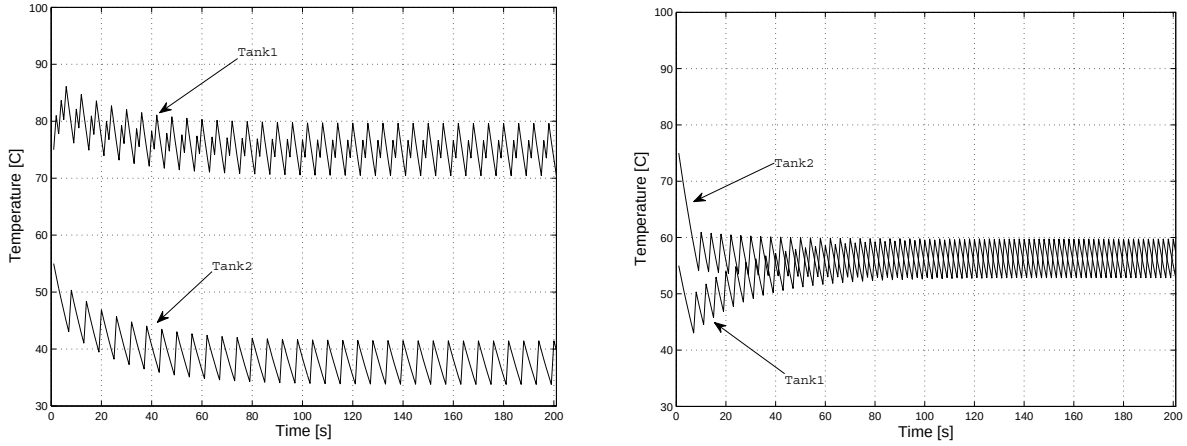


Figure 5: Principle for mapping temperature changes to time when the temperature is decreasing (a) or increasing (b). We obtain a lower and an upper bound on time for changing temperature range.



(a) (b)

Figure 6: Simulation results with two observable ranges, one simulation for each control objective.

```
const int TEMP_INC[temperature_t] = { 0, 6, 7, 0 };
const int TEMP_DEC[temperature_t] = { 0, 18, 10, 0 };
const int GUARD_INC[temperature_t] = { 0, 2, 2, 0 };
const int GUARD_DEC[temperature_t] = { 0, 7, 3, 0 };
```

The system is initialised with tank 1 at 55 degrees and tank 2 at 75 degrees, which corresponds to `temp` being 1 and 2. We note that the model detects changes of temperature so the actual range of temperature depends on the state (heating or not). We now consder the following control objectives:

```
control: A[] temp1>=LOW && temp1<=HIGH && temp2>=LOW  && temp2<=HIGH
control: A[] temp1>=LOW && temp1<=LOW  && temp2>=LOW  && temp2<=HIGH
control: A[] temp1>=LOW && temp1<=HIGH && temp2>=HIGH && temp2<=HIGH
```

The two first objectives are enforcable and UPPAAL-TIGA generates strategies that we insert in SIMULINK. The third one is not. Figure 6 provides the result of the simulations in SIMULINK for the first (a) and second (b) properties.

References

[1] Patricia Bouyer, Kim G. Larsen aqnd Nicolas Markey, Ocan Sankur, and Claus Thrane. Timed automata can always be made implementable. 2011. To appear in Proceedings of CONCUR 2011.

[2] Alexandre David, Jacob Deleuran Grunnet, Jan Jakob Jessen, Kim G. Larsen, and Jacob Illum. Application of model-checking technology to controller synthesis. 2011. To appear in Proceedings of FMCO 2011.

[3] Rémi Jaubert and Pierre-Alain Reynier. Quantitative robustness analysis of flat timed automata. In Martin Hofmann, editor, *FOSSACS*, volume 6604 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2011.

[4] Nicolas Markey. Robustness in real-time systems. 2011. To appear in Proceedins of 6th IEEE International Symnposium on Industrial Embedded Systems.