INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

"D4.2.5 – Prototype for Multi-mode DAE Systems in OpenModelica"

"Sub WP 4.2: Prototypes of tool support for multi-mode DAE systems"

"Work Package 4: Systems with multiple operating modes"

MODRIO (11004)

Version 1.0

Date 2016-04-22

Authors

Bernhard Thiele LiU

Executive Summary

In this deliverable the semantics of Modelica 3.3 language elements for describing discrete-time synchronous state machines are extended with the ability to have continuous-time models as "states", which enables modeling of multi-mode DAE (Differential Algebraic Equation) systems. The extension has been prototypically implemented in the OpenModelica tool.

1. Overview and Introduction

Improving the SotA for multi-mode DAE systems has been one important goal within the MODRIO project. Technical systems often comprise multiple operation modes, e.g., if systems proceed from a startup mode to a nominal operation mode and from a nominal operation mode into a shutdown mode, or when switches between different nominal operation modes and failure modes occur. The behavior of each mode is defined by its own DAE system. Besides mathematical and numerical aspects it is of interest to provide modeling language elements that allow a convenient and clear description of such systems.

In this deliverable the semantics of Modelica 3.3 language elements for describing discrete-time synchronous state machines are extended with the ability to have continuous-time models as “states”. The extension has been prototypically implemented for the OpenModelica tool.

2. Implementation

The implementation extends the semantics of Modelica 3.3 state machines with support for differential equations within state components and state-events for triggering transition between modes (“states”). The implementation builds on the symbolic representation for Modelica state machines which was presented by the authors in [1]. Leveraging this representation facilitated sharing code between the clocked discrete-time Modelica state machine implementation and the continuous-time extension described in this report.

This implementation builds on the pioneering work regarding Modelica extensions for multi-mode systems conducted within the MODRIO project, particularly the proposal by Elmqvist, Mattsson, and Otter for continuous-time state machines with causal blocks [2]. These kind of state machines resemble a well-known approach in automata theory, called hybrid automata, see e.g., [3].

The translation process is depicted in Figure 1. State machines are first flattened in the compiler front-end according to the approach outlined in [1]. After that, the state machine structures are further elaborated in the back-end where they are translated to basic hybrid data-flow equations (by transforming the abstract syntax tree (AST)).

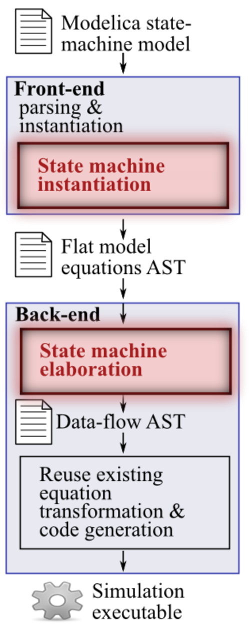

Modelica state-machine model
Front-end
parsing & instantiation
State machine instantiation
Flat model equations AST
Back-end
State machine elaboration
Data-flow AST
Reuse existing equation transformation & code generation
Simulation executable

Figure 1. Outline of the of the state machine compilation process [1].

Hybrid automata are a well-known modeling paradigm in automata theory, see e.g., [3]. In this section the hybrid automaton example given in [2] (see Figure 2) is discussed and the slightly different design decisions which have been explored in OpenModelica are highlighted.

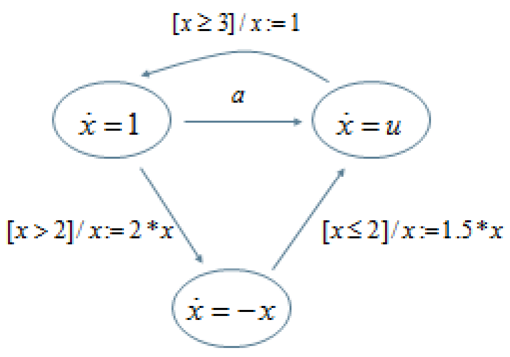


Figure 2. Hybrid automaton with "a" as input signal [2].

Every "state" in Figure 2 contains an ODE with a variable "x" which is shared between the states. Transition consists of trigger conditions when to switch the state (e.g., "[x>2]") and reset statements for the state of the ODE (e.g., "x := 2*x") which define the value to which the continuous-time state x is reset before entering the target "state".

For the OpenModelica prototype we explored a semantic extension for the discrete-time Modelica state machine that supports a comparable modeling style. Figure 4 shows our Modelica model of the hybrid automaton from Figure 2. The Dymola tool editor was used for creating the graphical view since the graphical OpenModelica editor OMEdit does not yet support the rendering of state machines (one needs to work on the textual editing level when working with the OpenModelica prototype for state machines). The result from simulating our Modelica version of the hybrid automaton with our OpenModelica prototype is shown in Figure 3. To enable the continuous-time state machine prototype extension, the simulation flag "--ctStateMachines" needs to be set.

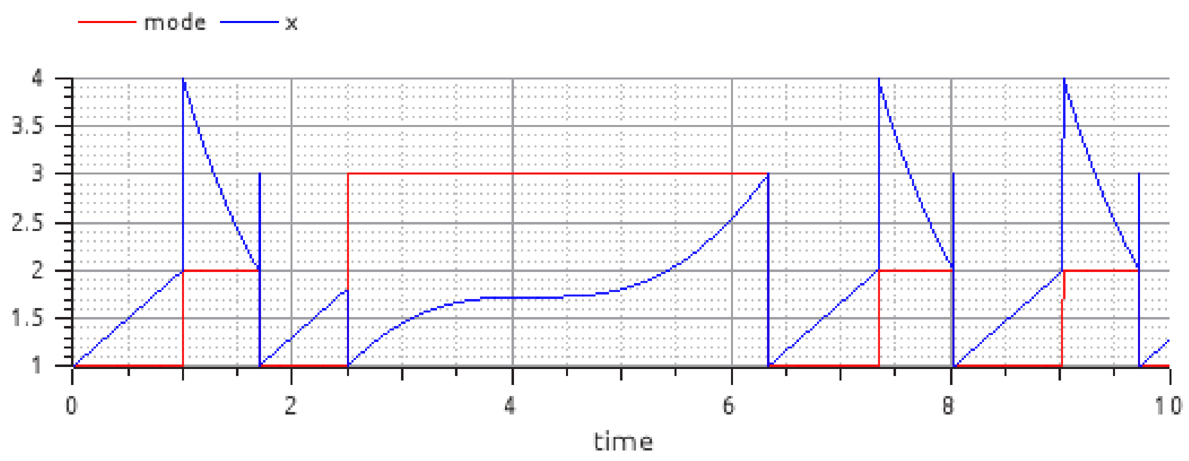
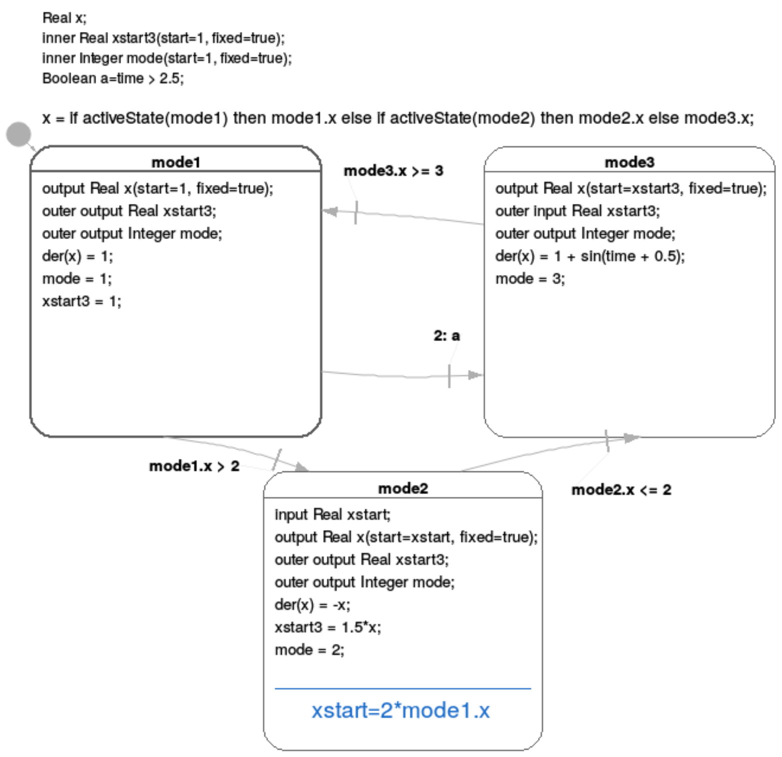


Figure 3. Simulation result of the hybrid automaton example (see Figure 2 and 4) using the OpenModelica prototype implementation and "a = time>2.5".

Different to the prototype presented in [2], variable "x" is not shared between the three "states", instead it is defined locally in each mode and an additional equation is introduced at the outer scope to merge the locally defined variables ("x = **if** **activeState**(mode1) **then** mode1.x **else** **if** **activeState**(mode2) **then** mode2.x **else** mode3.x;"). Discrete-time Modelica state machines can be entered by "reset" transitions with the semantics that the discrete-time states defined in the target states are reset to their "start"-value. In our prototype this semantics has been extended to continuous-time states. Thus, the semantics between discrete-time and continuous-time states is more closely maintained than in the prototype presented in [2], but an additional explicit merging equation is required. Similarly to [2], the "start" attribute is allowed to be bound to a variable expression (which is non-standard Modelica, since Modelica only allows parametric variability).

The reset conditions in the example are given either as constant ("mode1"), as input binding equation ("mode2") or they are communicated using the shared "global" variable "xstart3" ("mode3"). The sharing of variables is achieved by using "inner/outer" constructs which is identical to the approach used in discrete-time Modelica state machines.



```
Real x;
inner Real xstart3(start=1, fixed=true);
inner Integer mode(start=1, fixed=true);
Boolean a=time > 2.5;

x = if activeState(mode1) then mode1.x else if activeState(mode2) then mode2.x else mode3.x;
```

```
mode1
output Real x(start=1, fixed=true);
outer output Real xstart3;
outer output Integer mode;
der(x) = 1;
mode = 1;
xstart3 = 1;
```

```
mode3
output Real x(start=xstart3, fixed=true);
outer input Real xstart3;
outer output Integer mode;
der(x) = 1 + sin(time + 0.5);
mode = 3;
```

```
mode2
input Real xstart;
output Real x(start=xstart, fixed=true);
outer output Real xstart3;
outer output Integer mode;
der(x) = -x;
xstart3 = 1.5*x;
mode = 2;
```

```
model CT_HybridAutomaton
  Real x;
  inner Real xstart3(start=1, fixed=true);
  inner Integer mode(start=1, fixed=true);
  Boolean a = time > 2.5;
  block Mode1
    output Real x(start=1, fixed=true);
    outer output Real xstart3;
    outer output Integer mode;
  equation
    der(x) = 1;
    mode = 1;
    xstart3 = 1;
  end Mode1;
  Mode1 mode1;

  block Mode2
    input Real xstart;
    output Real x(start=xstart, fixed=true);
    outer output Real xstart3;
    outer output Integer mode;
  equation
    der(x) = -x;
    xstart3 = 1.5*x;
    mode = 2;
  end Mode2;
  Mode2 mode2(xstart=2*mode1.x);

  block Mode3
    output Real x(start=xstart3, fixed=true);
    outer input Real xstart3;
    outer output Integer mode;
  equation
    der(x) = 1 + sin(time + 0.5);
    mode = 3;
  end Mode3;
  Mode3 mode3;

equation
  x = if activeState(mode1) then mode1.x else
    if activeState(mode2) then mode2.x else mode3.x;
  initialState(mode1);
  transition(mode1, mode2, mode1.x > 2, priority=1);
  transition(mode1, mode3, a, priority=2);
  transition(mode3, mode1, mode3.x >= 3);
  transition(mode2, mode3, mode2.x <= 2);
end CT_HybridAutomaton;
```

Figure 4. Modelica model corresponding to the hybrid automaton from Figure 2 (graphical component view and textual view without graphical annotation).

3. Conclusion

A prototype for multi-mode DAE systems in OpenModelica has been implemented. For this means the semantics of Modelica 3.3 language elements for describing discrete-time synchronous state machines have been extended with the ability to have continuous-time models as "states". The implementation is motivated by pioneering work regarding Modelica extensions for multi-mode systems conducted within the MODRIO project and has been tested using several examples.

4. References

[1] Thiele, B., Pop, A., and Fritzson, P. Flattening of Modelica State Machines: A Practical Symbolic Representation. In Proc. of 11th International Modelica Conference. September 21-23, 2015, Versailles, France. https://doi.org/10.3384/ecp15118255

[2] Elmqvist, H., Matson, S.E, and Otter, M. Modelica extensions for Multi-Mode DAE Systems. In Proc. of 10th International Modelica Conference. March 10-12, 2014, Lund, Sweden. https://doi.org/10.3384/ECP14096183

[3] Henzinger, T.A. The theory of hybrid automata. In Proc. of 11th Annual IEEE Symposium on Logic in Computer Science (LICS), 1996.