The use of the UML within the modelling process of Modelica-models

Christoph Nytsch-Geusen[1]

[1] Fraunhofer Institute for Computer Architecture and Software Technology,
Kekuléstr. 7, 12489 Berlin, Germany
christoph.nytsch@first.fraunhofe.de

Abstract. This paper presents the use of the Unified Modeling Language (UML) in the context of object-oriented modelling and simulation of hybrid systems with Modelica. The definition of a specialized version of UML for the graphical description and model based development of hybrid systems in Modelica – the UML^H - was taken place in the GENSIM project [1, 2]. For a better support of the modelling process, an UML^H editor with different views (class diagrams, statechart diagrams, collaboration diagrams) was implemented as a part of the Modelica simulation tool MOSILAB [3]. In the EOOLT-workshop the use of UML^H and its semantics will be demonstrated by the development of a simplified model of a Pool-Billiard game in Modelica.

Keywords: UML^H, modelling of hybrid systems, Modelica

1 Introduction

On the one hand, the Unified Modeling language (UML) is the established standard for the development and graphical description of object-oriented software systems [4]. The UML offers a couple of diagrams, which describe different views (e.g. class diagrams, statechart diagrams, collaboration diagrams) on object-oriented classes. On the other hand *Modelica* [5] is a pure textual simulation language, which means the program code of long and highly structured models might be often heavy to understand. Thus, the combination of UML and Modelica was taken place within the GENSIM project. An UML editor for the Modelica based simulation tool MOSILAB was developed, which can be used for describing and generating Modelica models in a graphical way [3].

In this paper a special forming of UML for the modelling process of hybrid systems, the UML^H, will be presented. In a first step, the elements of he UML^H and their semantics for the Modelica-language will be introduced. After that, the use of UML^H will be illustrated by the example of a simplified version of a Pool-Billiard game.

2 UML^H and Modelica

The development of the UML^H was motivated by the following main reasons:

- to support the user within the modelling process of complex Modelica models in a easy manner,
- to have a method for the graphical documentation of the object-oriented construction of Modelica-models,
- to have a graphical analogy for the statechart extension of Modelica, which was introduced in the GENSIM project as a linguistic means of expression for model structural dynamics.

The UML^H includes only a subset of the UML standard, which is necessary for the graphical description of Modelica models: the class diagram view, the statechart diagram view and the collaboration diagram view.

2.1 Class-diagrams

A class diagram in UML^H is a rectangle, which contains in the upper part the class name and the Modelica class type. The optional lower part comprises the attributes (parameters, variables etc.) of the Modelica class. Inheritance and composition is expressed in the same way as in UML (compare with Fig. 1.)

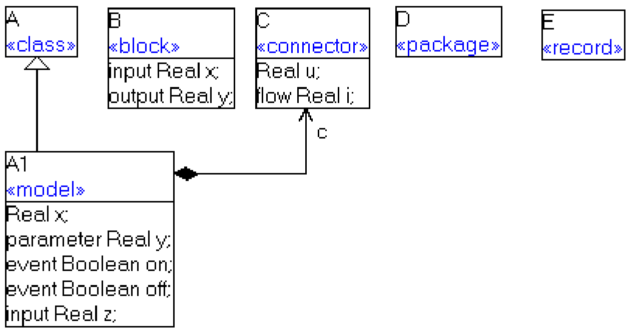

Fig. 1. UML^H class diagram

Starting from this graphical notation, the correspondent Modelica code can be generated automatically, e.g. with MOSILAB[1]. The following code shows the classes *A*, *A1* and *C*, which are inner classes of the package *UML_H*:

[1] In MOSILAB the UML^H diagrams are directly integrated within the Modelica code by the use of specialized annotations.

```
package UML_H
  annotation(UMLH(ClassDiagram="<umlhclass><name>…);

  class A
    annotation(UMLH(classPos=[31,53]));
  end A;

  model A1
    annotation(Icon(Text(extent=…,string="A1", …));
    annotation(UMLH(classPos=[31,146]));
    extends A;
    event Boolean on;
    event Boolean off;
    Real x;
    input Real z;
    parameter Real y;
    C c;
  ...
  end A1;

  connector C annotation(UMLH(classPos=[192,54]));
    Real u;
    flow Real i;
  end C;
...
end UML_H;
```

2.2 Collaboration diagrams

Collaboration diagrams in UML^H are also rectangles, which contain the object name and the type or the icon of the Modelica class, divided by a horizontal line. Four different connections types exist between the objects (see with Fig. 2.):

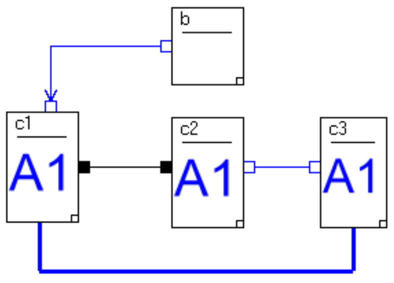

Fig. 2. UML^H collaboration diagram.

- Type 1: connections of connector variables (thin black line with filled squares at the ends)

- Type 2: connections of scalar variables (thin blue line with unfilled squares at the ends)
- Type 3: connections of scalar input/output variables (thin blue line with an arrow and a unfilled square)
- Type 4: multi-connections as a mixture of different connection types, e.g. type 1 and type 2 (fat blue line)

The following Modelica-code expresses the collaboration-diagram of Fig. 2:

```
model System
  annotation(CompConnectors(CompConn(label="label2",
             points=[-81,52; -81,43; -24,43; -24,51])));
  UML_H.A1 c1 annotation(extent=[-87,72; -74,52]);
  UML_H.A1 c2 annotation(extent=[-57,71; -44,51]);
  UML_H.A1 c3 annotation(extent=[-30,71; -18,51]);
  UML_H.B b annotation(extent=[-57,91; -44,77]);
equation
    // connection type 1:
    connect(c1.c,c2.c)annotation(points=[-74,62;-57,62]);
    // connection type 2:
    c2.y=c3.y annotation(points=[-44,62; -30,62]);
    // connection type 4 (mixture of type 1 and 2):
    connect(c1.c,c3.c) annotation(label="label2");
    c1.x=c3.x annotation(label="label2");
    // connection type 3:
    b.y=c1.z annotation(points=[-57,84; -79,84; -79,72]);
end System;
```

2.3 Statechart diagrams

A statechart diagram in UML^H is represented as a rectangle with round corners. In general, a statechart diagram contains several states and the transition definition between the states. Figure 3 shows four different types of States:

- Initial states, symbolized with a black filled circle,
- Final states, symbolized with a point in a unfilled circle,
- Atomic states, with a flat internal structure,
- Normal states, which can contain additional entry or exit actions and can be sub-structured in further statechart diagrams.

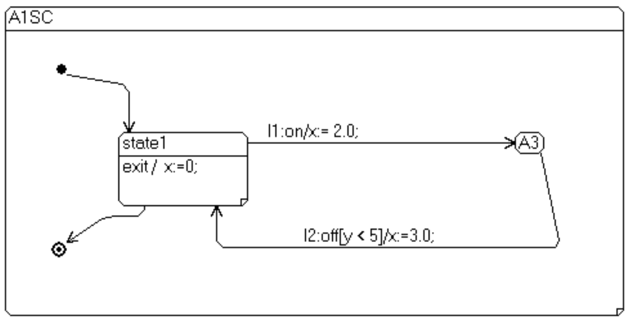

Fig. 3. UML^{H} statechart diagram.

The transitions between the states are specified with an optional label, an event, an optional guard and the action part. The following code shows the corresponding code of the statechart section[2] of the model A1:

```
model A1
...
statechart
  state A1SC extends State
    annotation(extent=[-88,86; 32,27]);
    state State1
      extends State;
      exit action x:=0; end exit;
    end State1;
    State1 state1 annotation(extent=[-66,62; -41,48]);
    State A3 annotation(extent=...);
    State I5(isInitial=true)...;
    State F7(isFinal=true)...;
    transition I5->state1 end transition
      annotation(points=[-76,73;-64,71; -64,62]);

    transition l1:state1->A3 event on action x:= 2.0;
    end transition annotation(points=...);

    transition l2:A3->state1 event off guard y < 5
      action x:=3.0;
    end transition ...;

    transition state1->F7 end transition annotation...;
   end A1SC;
  end A1;
```

[2] The new introduced statechart section is part of the Modelica language extension for model structural dynamics [6].

3 Example for UML^H-modeling

The modelling and simulation of a simplified Pool-Billiard game shall demonstrate the advantages of the graphical modelling with UML^H.

3.1 Model of a Pool-Billiard game

The system model of the Pool-Billiard game includes sub models for the balls and the table. The configuration of the system model is illustrated in Fig. 4. Following simplifications are assumed in the model:

- The Pool-Billiard game knows only a black, a white and a coloured ball.
- The table has only one hole instead of 6 holes.
- The collision-model is strong simplified.
- The balls are moving between the collisions and reflections only on straight directions in the dimension x and y.
- The reflections on the borders take place ideal without any friction losses.
- The rolling balls are slowed down with a linear friction coefficient f_r:

$$m \cdot \frac{dv_x}{dt} = -v_x \cdot f_r \text{ and } \frac{dx}{dt} = v_x \qquad \textbf{(1)}$$

$$m \cdot \frac{dv_y}{dt} = -v_y \cdot f_r \text{ and } \frac{dy}{dt} = v_y \qquad \textbf{(2)}$$

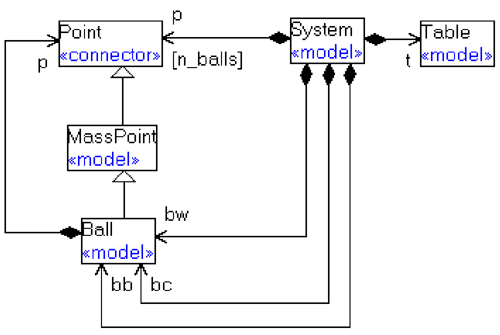

Fig. 4. UML^H class diagram of the Pool-Billard model

Fig. 5 shows the statechart diagram for the ball model. After the model enter the state *Rolling*, the ball knows four reflection events, for the four different borders of the billiard table. Depending from the border event, the new initial conditions (velocity and position) after the reflections are set and the ball enters again the state *Rolling*:

```
model Ball
  extends MassPoint(m=0.2);
  parameter SIunits.Length width;
  parameter SIunits.Length length;
  parameter SIunits.Length d = 0.0572 "diameter";
  parameter Real f_r = 0.1 "friction coefficient";
  SIunits.Velocity v_x, v_y;
  event Boolean reflection_left(start = false);
  ...
  equation
    reflection_left = if x < d/2.0;
    m * der(v_x)  = - v_x * f_r;
    der(x) = v_x;
    ...
  statechart
    state BallSC extends State;
      State Rolling;
      State startState(isInitial=true);
      ...
      transition startState -> Rolling
      end transition;
      ...
      transition Rolling->Rolling event reflection_left
        action v_x := -v_x; x := d/2.0;
      end transition;
      ...
    end BallSC;
  end Ball;
```

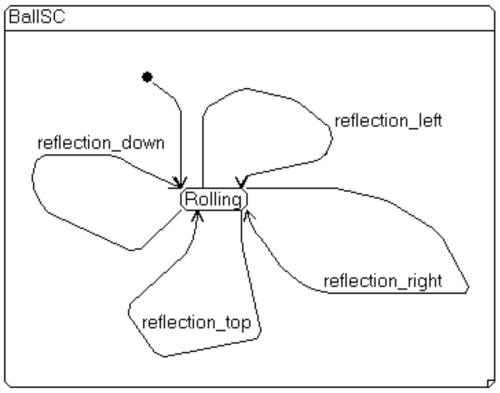

Fig. 5. UML^H statechart diagram of the ball model

On the system level two different states (*Playing* and *GameOver*) and two types of events - the collision of two balls and the disappearance of a ball in the hole (compare with Fig. 6 and the program code) exist. If the white ball enters the hole, the game will be continued with the white ball from the starting point (transition from *Playing* to *Playing*). If the black ball disappears in the hole, the statechart is triggered to the state *GameOver*. If the coloured ball disappeared, the game is reduced for one ball - *remove(bc)* - and the numerical calculation will be continued with a smaller equation system[3]:

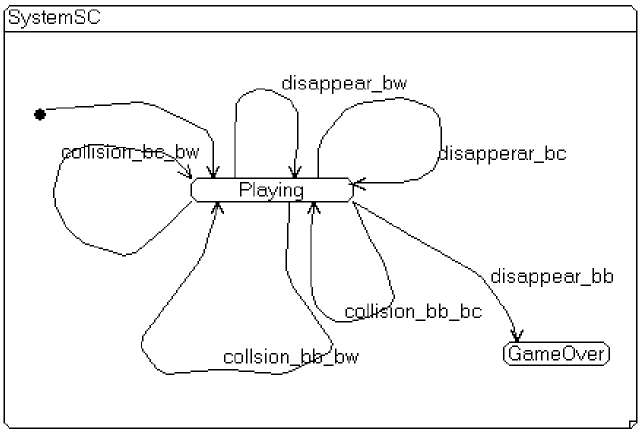

Fig. 6. UML^H-statechart-diagram for the model

```
model System
  parameter SIunits.Length d_balls = 0.0572;
  parameter SIunits.Length d_holes = 0.15;
  dynamic Ball bw, bb, bc; //structural dynamic submodels
  Table t(width = 1.27, length = 2.54);
  event Boolean disappear_bw(start = false);
  event Boolean disappear_bb(start = false);
  event Boolean disappear_bc(start = false);
  event Boolean collision_bw_bb(start = false);
  ...
  event Boolean push(start = false);

equation
  push = if fabs(bw.v_x)<0.005 and fabs(bw.v_y) < 0.005;
  disappear_bw = if((p[1].x-0)^2+(p[1].y-0)^2)^0.5
                   < d_holes;
  collision_bw_bb = if((p[2].x-p[1].x)^2
                   +(p[2].y-p[1].y)^2)^0.5 < d_balls;
  ...
statechart
  state SystemSC extends State;
    State Playing, startState(isInitial=true), GameOver;
    ...
    transition startState -> Playing action
      bw := new Ball(d = d_balls,...); add(bw);
```

[3] This model reduction mechanism takes place by using the model structural dynamics from MOSILAB [1].

```
      bb := new Ball(...); add(bb);
      bc := new Ball(...); add(bc);
    end transition;

    transition Playing->Playing event disappear_bw action
      ...
      remove(bw);
      bw := new Ball(x(start=1.27/2.9), y(start=0.6));
    end transition;

    transition Playing->Playing event disappear_bc action
      ...
      remove(bb);
    end transition;

    transition Playing -> GameOver event disappear_bb
    end transition;

    transition Playing->Playing event collision_bw_bb
      action
      v_x := bw.v_x; v_y := bw.v_y;
      bw.v_x := bb.v_x; bw.v_y := bb.v_y;
      bb.v_x := v_x; bb.v_y := v_y;
    end transition;
  end SystemSC;
end System;
```

3.1 Simulation experiment

The following simulation experiment illustrates the previous explained behaviour of the Pool-Billiard game. The parameter of the model are set in a manner, that all different types of events (1: collision of two balls, 2: reflection on a border, 3: disappearing in the hole) are present during the simulation experiment (see Fig. 7).

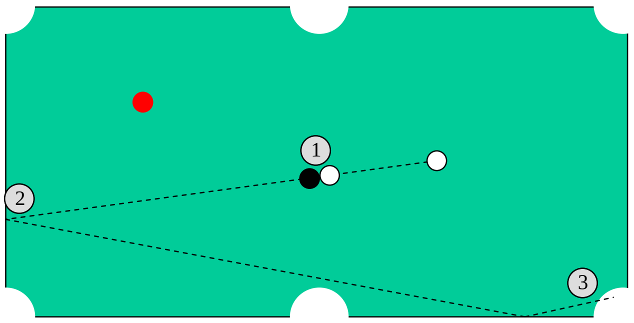

Fig. 7. Event types in the Pool-Billiard game

Figure 8 show the positions and the Figures 9 and 10 the reflection and collision events of the white and the black ball during a simulation period of 4 seconds.

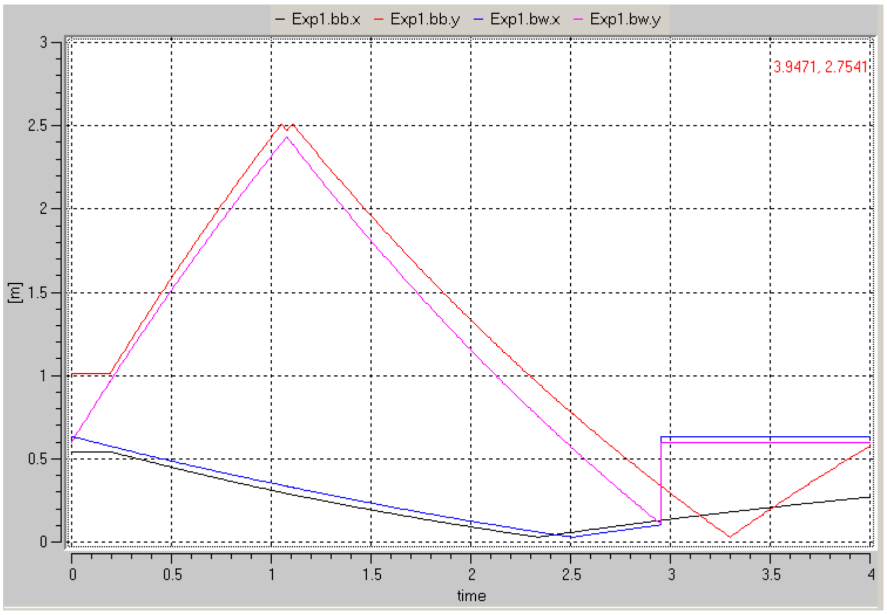

Fig. 8. x- and y-positions of the white and the black ball

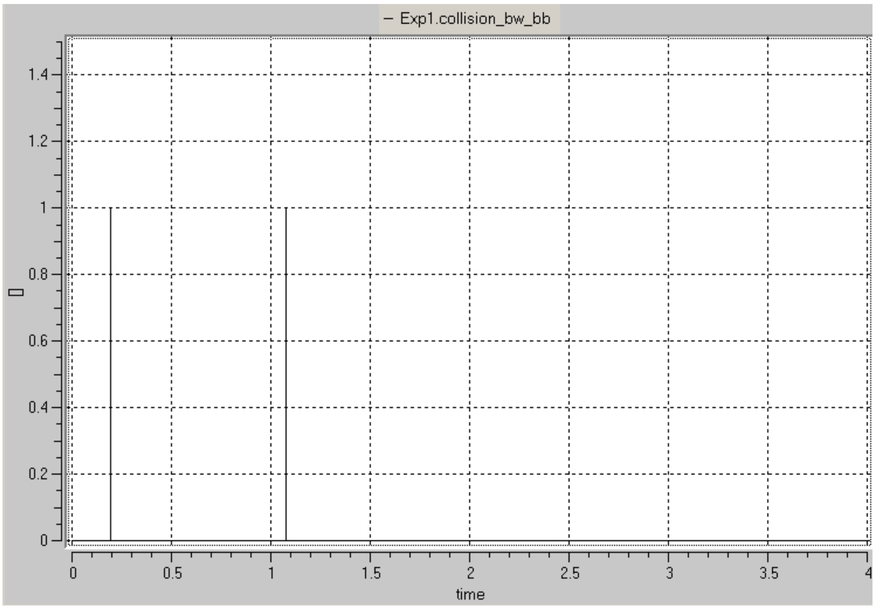

Fig. 9. Collision events of the white and the black ball

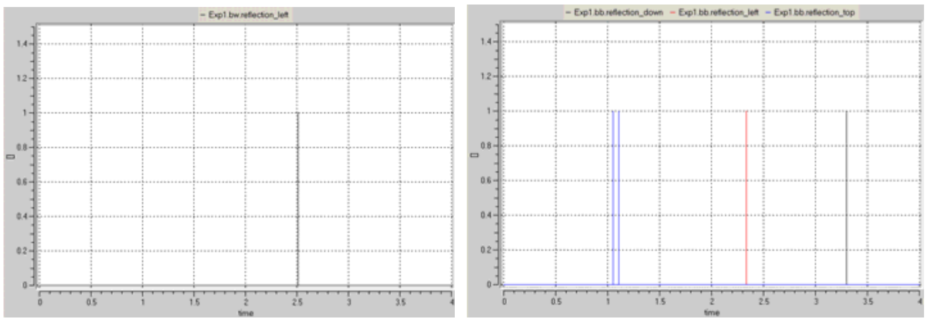

Fig. 10. Reflection events of the white ball (left) and the black ball (right)

After 0.2 seconds, the white ball collides with the black ball. After 1 second, the black ball is reflected twice in a short time period on the top side on the billiard-table and both balls collide again between its reflections. After 2.3 and 2.5 seconds the balls reflect on the left border. At 2.95 seconds the white ball drops into the hole. At the end, the white ball is set again on its starting position.

4 Conclusions

The example of the modelling and simulation of a Pool-Billiard game has shown the advantages of the graphical modelling with UML^H for Modelica models. With UML^H, the design of a complex system model in Modelica begins with the drawing of its model structure. The class diagrams und the collaboration diagrams describe the object-oriented model construction and the statechart diagrams are used for the formulation of the event-driven model behaviour. If the Modelica tool supports code generation like MOSILAB, the Modelica code can be obtain automatically from the UML^H model. This pure code has to be filled up by the user with model equations (physical behaviour) of the modelled system.

References

1. Nytsch-Geusen, C. et al.: MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics. Proceedings of the 4th International Modelica Conference, TU Hamburg-Harburg, Hamburg, 2005.
2. Nytsch-Geusen, C. et al.: Advanced modeling and simulation techniques in MOSILAB: A system development case study. Proceedings of the 5th International Modelica Conference, Arsenal Research, Wien, 2006.
3. MOSILAB-Webportal: http://www.mosilab.de.
4. UML-Homepage: http://www.uml.org.
5. Modelica-Homepage: http: /www.modelica.org.
6. Nordwig, A. et al.: MOSILA-Modellbeschreibungssprache, Version 2.0, Fraunhofer Gesellschaft, 2006.