Design and validation of an annotation-concept for the representation of 3D-geometries in Modelica

Thomas Hoeft[1] Christoph Nytsch-Geusen[1, 2]
[1]Fraunhofer Institute for Computer Architecture and Software Technology
Kekuléstraße 7, 12489 Berlin, Germany
[2]University of Arts Berlin, Hardenbergstraße 33, 10623 Berlin, Germany
christoph.nytsch@first.fraunhofer.de

Abstract

Simulation models of complex technical systems need beside the description of their physical behaviors also a representation of their 3-dimensional geometry and topology. Up to now, the Modelica language specification [1] includes only rules for 2D-primitives in form of specialized annotations. Starting from this point, this paper illustrates the design and validation of an advanced annotation-concept for embedded 3D-geometries in physical models. The basic idea consists in the combination of specialized 3D-annotions for classes and objects with a standardized description of 3D-geometries and topologies. Therefore the X3D-standard [2] is used by the authors. Based on the founded similarities and parallelisms in the object-oriented concept of Modelica and the node concept of X3D an annotation concept for the embedding of the 3D-geometries was designed. Further an extension for the Modelica-simulator MOSILAB [3] in form of a 3D-editor plug-in was developed for the generation of X3D/Modelica-scenes and the validation of the new annotation concept. Finally the annotation-concept was evaluated in a simulation use case, where the physical model of a simplified Pool-Billiard game [4] was combined with its 3D-geometry description.

Keywords: 3D-annotation concept; X3D; 3D-representation of physical models; 4D-animation

1 Introduction

Up to now the Modelica language specification does not comprise means of expressions for code integrated description of 3D-geometries. The first fundamental analysis and conceptual work in this direction was done by [5]. Two alternative ways were discussed by the author for the integration of 3D object information in Modelica:

1. Definition of a basic set of “graphical classes”, which make a representation of primitive 3D objects (e.g. *Triangle*, *Sphere*) and position operations with this objects (e.g. *Translation*, *Rotation*) in user defined physical models possible.
2. Direct integration of the 3D object information as “graphical annotations” into the physical models self.

Further the embedding of external graphical formats like STL, VRML or DXF in Modelica models as annotations information was shortly discussed in this paper.

The Modelica-simulator Dymola [6] supports with an additional software component the visualization of 3D-objects, mainly for the MultiBody-Library. For this, external definitions of 3D-shapes via dxf-files are utilized.

In our approach for a model integrated representation of 3D objects, we have introduced *Extensible 3D Graphics* (X3D) - an open international standard for 3D on the Web and the official successor of VRML - into the Modelica language as a new annotation-type. We think this approach offers a number of important advantages:

- X3D represents a sophisticated (and international accepted) concept for complex and hierarchical structured 3D-scenes, which fits well to the object-oriented Modelica language concept.
- The prototype-concept of X3D allows an efficient integration in the object-oriented concept of Modelica.
- The annotation concept of Modelica supports the X3D integration by adding the 3D-geometrical information as X3D-strings on class level or object level. Modelica-tools, which don’t understand those X3D-annotations, are not bothered.
- The use of X3D in Modelica classes enables a simple export of the 3D representation of a physical model as a X3D-scene.

2 Annotation concept for 3D-object representation in Modelica

For the integration and validation of X3D in the Modelica language we have done following three steps:

1. **Design of an annotation concept for the representation of 3D-geometries in Modelica**
   This comprises
   - the definition of the language subset of X3D, which is necessary for the representation of 3D-objects in Modelica,
   - the definition of the annotation syntax for X3D information,
   - the Modelica class definition of a set of 3D-primitives as a base for complex 3D-scenes,
   - the rules to instantiate this 3D-classes in physical models and
   - a syntax for the coupling between the X3D object attributes and the Modelica variables for 3D-animated simulation experiments.
2. **Development of a 3D-editor for the generation and validation of Modelica models, which contain 3D-objects, described in X3D**
   This comprises
   - the definition of a set of 3D-base objects and their attributes, which shall be supported by the editor,
   - the design and the implementation of the construction interface for 3D-scenes and
   - the integration of the 3D-editor in the MOSILAB-IDE [3] as a plug-in.
3. **Evaluation of the annotation-concept with the help of a use case**
   The analyzed system model and its graphical representation
   - have to have a nontrivial recursive hierarchical structured geometry and
   - have to include static and animated subcomponents.

2.1 Graphical representation in X3D

The X3D specification uses a hierarchical node concept by the use of the XML-Syntax. A single node is described by its node type and a number of fields (node attributes). Each field has to be declared with one of the 26 X3D data types. The following simple X3D-scene, composed of a blue and a red ball, explains the main features of X3D for our use in the context with Modelica. In a first step, a reusable prototype *Ball* is defined with the *ProtoDeclare* node. The first subnode, named *ProtoInterface*, contains the field declarations, for which values can be set during the instantiation of the prototype:

```
<X3D profile="Immersive">
 <Scene>
  <ProtoDeclare name="Ball">
   <ProtoInterface>
    <field accessType="initializeOnly"
    name="radius" type="SFFloat" value="1.0"/>
    <field accessType="initializeOnly"
    name="diffuseColor" type="SFColor"
    value="0.8 0.8 0.8"/>
    <field accessType="initializeOnly"
    name="translation" type="SFVec3f"
    value="0.0 0.0 0.0"/>
    ...
   </ProtoInterface>
```

The second subnode, named *ProtoBody*, defines the functionality of the prototype. The three-dimensional geometry of the ball is described by the node for the X3D-primitive *Sphere* and its optical appearance (*diffuseColor*, *transparency* …) by the *Material*-node. The ball position and orientation is defined by the *Transform*-node with the fields *translation* and *rotation*. Further, the code snippet shows some connections between a *nodeField* of a subnode within the *ProtoBody*-node and a declared *protoField* of the *ProtoInterface*-node. This concept makes the access to these quantities possible during the instantiation of the prototype:

```
   <ProtoBody>
    <Transform>
     <IS>
      <connect nodeField="translation"
      protoField="translation"/>
      ...
     </IS>
     <Shape>
      <Sphere>
       <connect nodeField="radius"
       protoField="radius"/>
      </Sphere>
      <Appearance>
       <Material>
        <connect nodeField="diffuseColor"
        protoField="diffuseColor"/>
        ...
       </Material>
      </Appearance>
     </Shape>
    </Transform>
   </ProtoBody>
  </ProtoDeclare>
```

In the second step, the both objects *ballBlue* and *ballRed* with the prototype *Ball* are instantiated, whereas the *radius* value is set on the typical size for a billiard ball (2.65 cm) and the *diffuseColor* value is set on the RGB-values for blue and red. The red ball is displaced from the origin at 25 cm by setting the value of the *transform* field:

```
  <ProtoInstance name="Ball">
   <MetadataString name="ballBlue"/>
   <fieldValue name="diffuseColor"
   value="0.0 0.0 1.0"/>
   <fieldValue name="radius" value="0.0265"/>
  </ProtoInstance>

  <ProtoInstance name="Ball">
    <MetadataString name="ballRed"/>
    <fieldValue name="diffuseColor"
    value="1.0 0.0 0.0"/>
    <fieldValue name="radius" value="0.0265"/>
    <fieldValue name="translation"
    value="0.25 0.0 0.0"/>
   </ProtoInstance>
 </Scene>
</X3D>
```

Figure 1 shows the visualization of this short X3D-scene. As the example illustrates, X3D has not a real object-oriented concept, but the *ProtoDeclare*-node with its *ProtoInterface* and *ProtoBody* subnodes has strong parallelism to the object composition in Modelica.

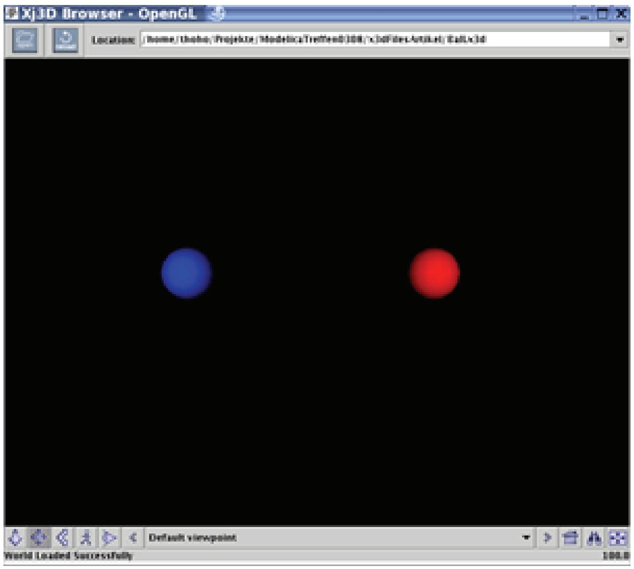
Xj3D Browser - OpenGL
Location: /home/thoho/Projekte/ModelicaTreffen0308/x3dFilesArtikel/Ball.x3d
Default viewpoint
World Loaded Successfully
100.0

Figure 1: Simple X3D-scene with two balls

2.2 Physical behavior in Modelica

The Modelica model of the ball describes its physical behavior with simplified equations of motion of a concentrated mass. Up to now, the model has not a representation of its three-dimensional geometry:

```
import Modelica.SIunits;
...
model Ball
  parameter SIunits.Mass m = 0.2;
  parameter Real f_r = 0.05 "friction coeffient";
  SIunits.Length x, y;
  SIunits.Velocity v_x, v_y;
equation
  m * der(v_x) = - v_x * f_r; der(x) = v_x;
  m * der(v_y) = - v_y * f_r; der(y) = v_y;
end Ball;
```

2.3 Integration of X3D in Modelica

Annotations in Modelica can be used as containers for additional information, which have no influence on the modeled physical behavior of a model class. Well known examples are the definitions of graphical 2D-objects for the model icons or the model documentation in form of embedded html-Code.

In our concept we have defined a new type of annotations, which contains parts of X3D-scenes as strings and give a Modelica-model a representation of its 3D-geometry. These annotations are labeled by a new element, named *Object3D* and can be used for classes and objects:

Use in the class context:

```
model ClassName
  annotation(Object3D(x3d="X3D-String"))
  ...
end ClassName;
```

Use in the object context:

```
model ClassName
  ...
  ClassType objectname
    annotation(Object3D(x3d="X3D-String"));
  ...
end ClassName;
```

At first, based on this syntax, we have defined a set of Modelica basic types for the 3D-modeling in the package *BasicBodies*:

- Sphere3D,
- Cone3D,
- Box3D,
- Cylinder3D,
- Point3D,
- PolyLine3D.

As an example, the following code shows the implementation of the basic type Sphere 3D:

```
package BasicBodies
  model Sphere3D annotation(Object3D(x3d = "
    <ProtoDeclare name=\" Sphere3D\">
     <ProtoInterface>
      <field accessType=\" initializeOnly\"
      name=\" radius\" type=\" SFFloat\"
      value=\" 1.0\"/>
      <field accessType=\" initializeOnly\"
      name=\" transparency\" type=\" SFFloat\"
      value=\" 0.0\"/>
      <field accessType=\" initializeOnly\"
      name=\" diffuseColor\" type=\" SFColor\"
      value=\" 0.8 0.8 0.8\"/>
      <field accessType=\" initializeOnly\"
      name=\" translation\" type=\" SFVec3f\"
      value=\" 0.0 0.0 0.0\"/>
      <field accessType=\" initializeOnly\"
      name=\" rotation\" type=\" SFRotation\"
      value=\" 0.0 0.0 1.0 1.0\"/>
     </ProtoInterface>
```

```
      <ProtoBody>
       <Transform>
        <IS>
         <connect nodeField=\" translation\"
         protoField=\" translation\"/>
         <connect nodeField=\" rotation\"
         protoField=\" rotation\"/>
        </IS>
        <Shape>
         <Sphere>
          <connect nodeField=\" radius\"
           protoField=\" radius\"/>
          </Sphere>
          <Appearance>
           <Material>
            <connect nodeField=\" diffuseColor\"
             protoField=\" diffuseColor\"/>
            <connect nodeField=\" transparency\"
             protoField=\" transparency\"/>
           </Material>
          </Appearance>
        </Shape>
       </Transform>
      </ProtoBody>
    </ProtoDeclare>"));
  end Sphere3D;
end BasicBodies;
```

The other basic 3D-types are described in a similar manner. Starting from these basic types, the configuration of complex 3D-models in Modelica can take place.

2.4 Coupling of the physical and geometrical model description

The decisive connection between the variables of the physical model and field-values of its X3D-representation is realized by the introduction of the annotation-element *coupling*. The syntax is defined as follows:

```
model ClassName
  annotation(Object3D(x3d="X3D-String”,
  coupling(protoFieldName1={v1,v2,0.0},
           protoFieldName2={v3}, ...)))
  ...
end ClassName;
```

At this, *protoFieldName* stands for the field in the 3D-representation, which shall be updated dynamically during the simulation (e.g. the object position or its size or color) and *v1, v2, v3* the corresponding Modelica variables. Thus, a physical model can have a number of coupled protoFields.

The next code piece shall illustrate this coupling concept with the help of the ball example in paragraphs 2.2 and 2.3. For this purpose, the *ProtoInterface* definition of the X3D description is integrated as an annotation on the class level, because this information concerns only the class interface. The *ProtoDeclare* node is omitted, because this information is implicit contained in the Modelica class-name itself:

```
...
import BasicBodies3D;
...
model Ball annotation(Object3D(x3d="
  <ProtoInterface>
   <field accessType=\" initializeOnly\"
   name=\" radius\" type=\" SFFloat\"
   value=\" 0.0265\"/>
   ...
   <field accessType="initializeOnly"
   name=\" translation\" type=\" SFVec3f\"
   value=\" 0.0 0.0 0.0\"/>
  </ProtoInterface>"),
  coupling(translation={x,y,0.0});
```

The representation of the 3D-geometry of the class *Ball* takes place by the instantiation of the basic 3D-type *Sphere3D* as an object within the class:

```
  BasicBodies3D.Sphere3D ball
  annotation(Object3D(x3d="
    <ProtoBody>
     <ProtoInstance
      name=\" BasicBodies3D.Sphere3D\">
      <MetadataString name=\" ball\"/>
      <connect nodeField=\" radius\"
      protoField=\" radius\"/>
      ...
      <connect nodeField=\" translation\"
      protoField=\" translation\"/>
     </ProtoInstance>
    </ProtoBody>")));
  parameter SIunits.Mass m = 0.2;
  parameter Real f_r = 0.05 “friction coeffient”;
  SIunits.Length x, y;
  SIunits.Velocity v_x, v_y;
equation
  m * der(v_x) = - v_x * f_r; der(x) = v_x;
  m * der(v_y) = - v_y * f_r; der(y) = v_y;
end Ball;
```

3 Use case Pool-Billard game for validating the annotation concept

In the use case, which shall validate our annotation concept for embedded 3D-geometry representations, we have used a model of a simplified Pool-Billiard game with three balls and one hole [4]. This simulation model suits well to the problem, because its geometry is hierarchical structured and includes static (table) and dynamic sub-components (billiard balls).

3.1 Modeling process

In the first step, a leg model (class *TableLeg*) from the billiard table shall be configured from the three submodels *bottom* (type *Cylinder3D*), *adapter* (type *Cone3D*) and *shaft* (type *Cylinder3D*):

```
import BasicBodies.*
...
model TableLeg annotation(Object3D(x3d="
  <ProtoInterface>
   <field accessType=\" initializeOnly\"
   name=\" bottom.height\" type=\" SFFloat\"
   value=\" 0.025\"/>
   <field accessType=\" initializeOnly\"
   name=\" bottom.radius\" type=\" SFFloat\"
   value=\" 0.125\"/>
   ...
   <field accessType=\" initializeOnly\"
   name=\" translation\" type=\" SFVec3f\"
   value=\" 0.0 0.0 0.0\"/>
   <field accessType=\" initializeOnly\"
   name=\" rotation\" type=\" SFRotation\"
   value=\" 0.0 0.0 1.0 0.0\"/>
  </ProtoInterface>"));

Cylinder3D bottom annotation(Object3D(x3d="
   <ProtoInstance name=\" Cylinder3D\">
    <MetadataString name=\" bottom\"/>
    <connect nodeField=\" radius\"
    protoField=\" bottom.radius\"/>
    <connect nodeField=\" height\"
    protoField=\" bottom.height\"/>
    ...
    <IS>
     <fieldValue name=\" translation\"
     value=\" 0.0 -0.4 0.0\"/>
    </IS>
   </ProtoInstance>"));

Cone3D adapter annotation(Object3D(x3d="
   <ProtoInstance name=\" Cone3D\">
    ...
   </ProtoInstance>"));

  Cylinder3D shaft annotation(Object3D(x3d="
   <ProtoInstance name=\" Cylinder3D\">
    ...
   </ProtoInstance>"));
end TableLeg;
```

Figure 2 shows the visualization of the previous defined 3D-representation of the *TableLeg* model class.

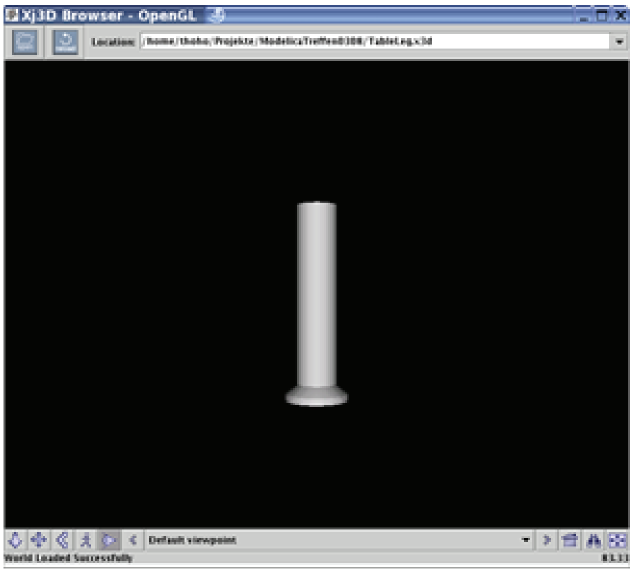

Figure 2: 3D-representation of the *TableLeg* model

On the next hierarchy level the submodels for the billiard table model are instantiated from two predefined model classes (*TableLeg*, *Border*) and from two 3D basic types classes (*Box3D*, *Cylinder3D*). The model class *BillardTable* includes the submodels for the plate, the borders, the legs and the hole. Figure 3 shows the visualization of this table model.

```
model BillardTable annotation(Object3D(x3d="
  <ProtoInterface>
   <field accessType=\" initializeOnly\"
   name=\" hole.height\" type=\" SFFloat\"
   value=\" 0.081\"/>
   <field accessType=\" initializeOnly\"
   name=\" hole.radius\" type=\" SFFloat\"
   value=\" 0.15\"/>
   ...
  </ProtoInterface>"));
  parameter SIunits.Length width,length;

  Box3D plate annotation(Object3D(x3d="
   <ProtoInstance name=\" Box3D\">
    <MetadataString name=\" plate\"/>
    <fieldValue name=\" size\"
    value=\" 2.54 0.08 1.27\"/>
    <fieldValue name=\" diffuseColor\"
    value=\" 0.0 1.0 0.0\"/>
    <fieldValue name=\" translation\"
    value=\" 0.0 0.0 0.0\"/>
   </ProtoInstance>"));

  Border borderUp annotation(Object3D(…));
  Border borderDown annotation(Object3D(…));
  Border borderLeft annotation(Object3D(…));
  Border borderRight annotation(Object3D(…));

  Cylinder3D hole annotation(Object3D(x3d="
   <ProtoInstance name=\" Cylinder3D\">
    ...
    <connect nodeField=\" height\"
    protoField=\" hole.height\"/>
    <connect nodeField=\" radius\"
    protoField=\" hole.radius\"/>
    <IS><fieldValue name=\" translation\"
    value=\" 1.26 0.0 -0.635\"/></IS>
   </ProtoInstance>"));

  TableLeg legDownLeft annotation(Object3D(x3d="
   <ProtoInstance name=\" TableLeg\">
    <MetadataString name=\" legDownLeft\"/>
    <fieldValue name=\" translation\"
    value=\" -1.06 -0.375 0.5\"/>
   </ProtoInstance>"));

  TableLeg legDownRight annotation(Object3D(…));
  TableLeg legUpLeft annotation(Object3D(…));
  TableLeg legUpRight annotation(Object3D(…));
end BillardTable;
```

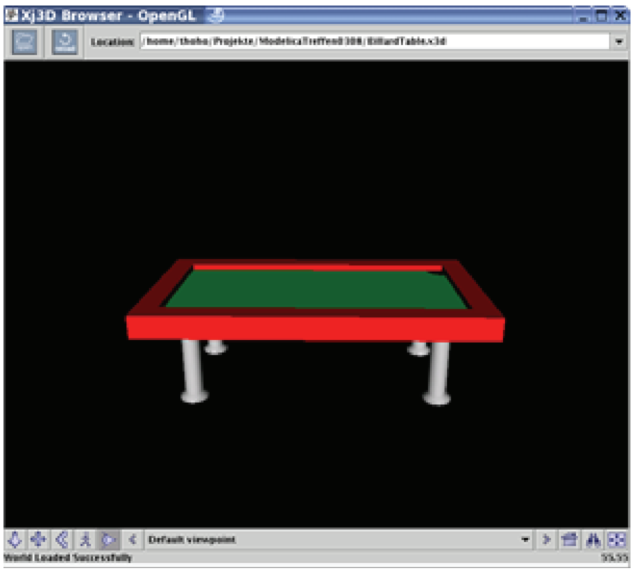

Figure 3: 3D-representation of the *BillardTable* model

The class *SystemModel* integrates the static table model and the three physical ball models. The physical model of the simplified Pool-Billiard game shall be drafted only roughly in this paper. A detailed description is given in [4]. The implementation of this example was realized with the language extension for Modelica for model structural dynamics from the GENSIM project [7, 8]. The different events of a billiard game (reflections, collisions) and a varying number of balls can be efficiently described with the concept of object-oriented statecharts and object dynamics:

```
...
model SystemModel
  annotation(Object3D(...));
  parameter Integer n_balls = 3;
  parameter Real v_x, v_y;
  parameter Real d_balls = 0.0572;
  parameter Real d_holes = 0.15;
  Point p[n_balls];

  dynamic Ball bw annotation(Object3D(x3d="
   <ProtoInstance name=\" Ball\">
    <MetadataString name=\" bw\"/>
    <fieldValue name=\" diffuseColor\"
    value=\" 1.0 1.0 1.0\"/>
    <fieldValue name=\" translation\"
    value=\" 0.8 0.066 -0.2\"/>
   </ProtoInstance>"));

  dynamic Ball bb annotation(Object3D(x3d="
   <ProtoInstance name=\" Ball\">
    <MetadataString name=\" bb\"/>
    <fieldValue name=\" diffuseColor\"
    value=\" 0.0 0.0 0.0\"/>
    <fieldValue name=\" translation\"
    value=\" 0.6 0.066 -0.2\"/>
  </ProtoInstance>"));

  dynamic Ball bc annotation(Object3D(x3d="
   <ProtoInstance name=\" Ball\">
    <MetadataString name=\" bc\"/>
    <fieldValue name=\" diffuseColor\"
    value=\" 0.0 0.0 1.0\"/>
    <fieldValue name=\" translation\"
    value=\" 0.4 0.066 -0.2\"/>
   </ProtoInstance>"));

  BillardTable t(width = 1.27, length = 2.54)
   annotation(Object3D(x3d="
   <ProtoInstance name=\" BillardTable\">
    <MetadataString name=\" t\"/>
   </ProtoInstance>"));

  event Boolean disappear_bw(start = false);
  event Boolean collision_bw_bb(start = false);
  ...
equation
  disappear_bw =
    if((p[1].x-0.0)^2+(p[1].y-0.0)^2)^0.5<d_holes
    then true else false;
  collision_bw_bb =
    if((p[2].x-p[1].x)^2+(p[2].y-p[1].y)^2)^0.5
       <d_balls then true else false;
  ...
statechart
  state SystemSC extends State;
    State startState(isInitial=true);
    State Playing, GameOver;
```

```
transition Playing->Playing
  event disappear_bw action
  disconnect(bw.p,p[1]); remove(bw);
  bw:=new Ball(d=d_balls, width=t.width,
               length = t.length,
               x(start = 1.27/2.0),
               y(start = 0.6));
  connect(bw.p,p[1]);
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 SystemModel;
```

Figure 4 illustrates the complete system model with the static and dynamic model parts.

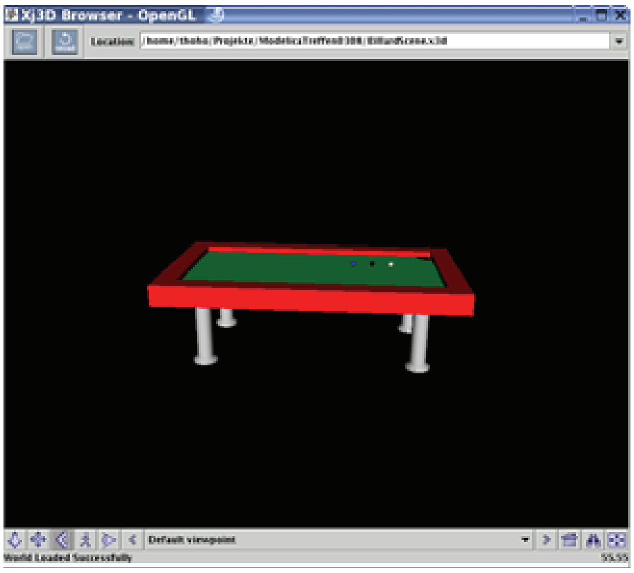

Figure 4: 3D-representation of the *SystemModel*

3.2 Simulation experiment

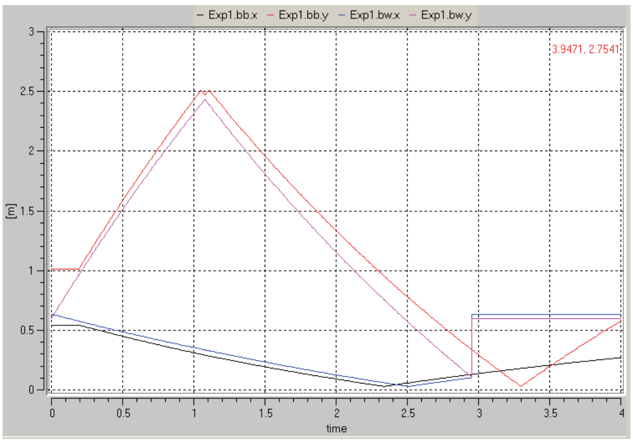

Figure 5: Simulation experiment for the Pool-Billiard game over 4 seconds.

Figure 5 shows the positions of the white and the black ball during a simulation period of 4 seconds.

After 0.2 seconds, the white ball collides with the black ball. After 1.0 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.

Figure 6 to Figure 9 show the 4D-animation of the same simulation experiment. Because the calculated x- and y-coordinates of both ball models are connected with their 3D-representations by the annotation-element *coupling* (compare with paragraph 2.4), a 4D-animation (3 space coordinates plus the time) of the experiment can be automatically generated.

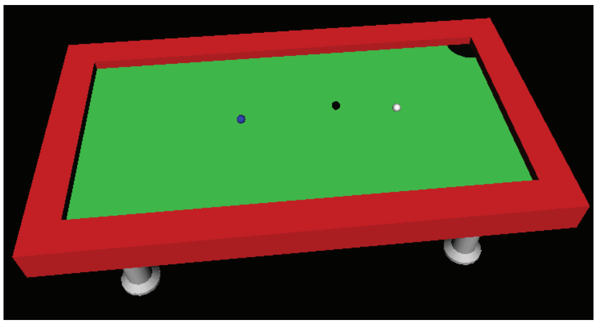

Figure 6: Simulation experiment at time=0 seconds

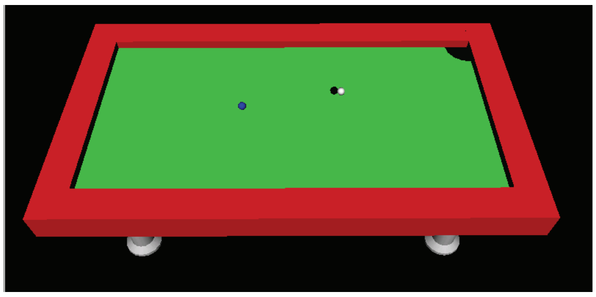

Figure 7: Simulation experiment at time=0.2 seconds

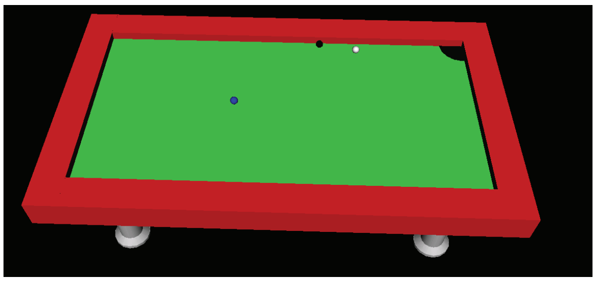

Figure 8: Simulation experiment at time=2.3 seconds

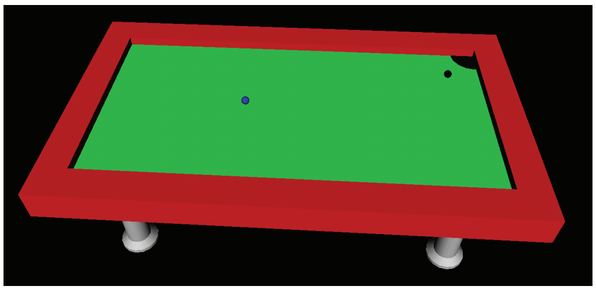

Figure 9:Simulation experiment at time=2.95 seconds

4 Development of a 3D-Model editor

For the validation of the previous described annotation concept for 3D-geometries in Modelica, a 3D-model editor is being developed by the authors. This editor supports the definition of 3D-scenes and generates the Modelica-code with the embedded X3D-description. Because the editor is implemented as a plug-in for the graphical user interface of the simulation tool MOSILAB [3], the 3D-modeling works close together with the other modeling features of the MOSILAB-IDE (compare with Figure 10).

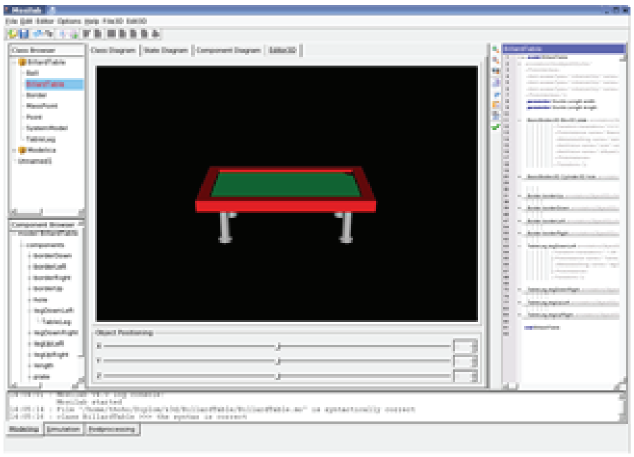

Figure 10: 3D-model editor as plug-in for the simulation tool MOSILAB .

5 Conclusions

The new designed annotation-concept for the representation of 3D-geometries in Modelica, based on the X3D-standard, offers a number of advantages and new perspectives:

- An integrated description of the equation based physical behavior and a corresponding representation of the 3D geometry in a unitary Modelica-model is an excellent precondition for an efficient communication between the physical and the geometrical model aspects.
- The use of X3D and its *ProtoDeclare*-concept fits well to the object-oriented concept of Modelica.
- 3D-objects, based on Modelica standard types, can be added directly to the physical models and could be connected by the coupling-annotation element.
- The modeling process of complex 3D-scenes can take place recursively on a multitude of hierarchical layers, because each Modelica/X3D-class can be reused on the next hierarchy-level (see

the modeling process of the billiard table in chapter 3.1).

- The use of the standard X3D-format for the modeling of the 3D-geometries within the Modelica language enables the import and export of 3D-scenes from/to X3D.
- Future works will focus on a closer integration of the 3D-editor in the simulation tool MOSILAB.

References

[1] Modelica Association Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.0, September 2007.

[2] Brutzman D. and Daly L. X3D: Extensible 3D Graphics for Web Authors. The Morgan Kaufmann Series in Interactive 3D Technology, 2007.

[3] MOSILAB-Homepage: http://www.mosilab.de

[4] Nytsch-Geusen C. The use of the UML within the modelling process of Modelica-models. EOOLT´2007, 1st International Workshop on Equation-Based Object-Oriented Languages and Tools, Berlin 2007.

[5] Engelson V. 3D Graphics and Modelica – an integrated approach. Linköping Electronic Articles in Computer and Information Science. Linköping universitet, 2000.

[6] Dymola-Homepage: http://www.dynasim.se

[7] 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.

[8] 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.