# THE MODELICA STANDARD LIBRARY AS AN ONTOLOGY FOR MODELING AND SIMULATION OF PHYSICAL SYSTEMS


| | Adrian Pop[1] | Peter Fritzson[2] | |
|---|---|---|---|

**Programming Environments Laboratory**
**Department of Computer and Information Science**
**Linköping University**
**58131 Linköping**
**Sweden**



## Abstract

This paper presents the Modelica Standard Library, an ontology used in modeling and simulation of physical systems. The Modelica Standard Library is continuously developed in the Modelica community. We present parts of the Modelica Standard Library and show an example of its usage. Also, in this paper we focus on the comparison of Modelica, the language used to specify the Modelica Standard Library with other ontology languages developed in the Semantic Web community.


## 1 Introduction and Related Work

The Modelica Standard Library provides concepts (classes) from various physical domains that can be easily used to create models (new classes). Also, these new created models can be further re-used.

As related work we can mention the PhySys ontology and OLMECO library [14] for dynamic physical systems. The PhySys ontology consists of three engineering ontologies formalizing conceptual viewpoints on physical systems: system layout, physical processes and descriptive mathematical relations. The PhySys ontology and the OLMECO library provide a similar framework as our work presented in [15] which is based on function-means decomposition of systems and Modelica components are associated with different means.

The paper is structured as follows: The next section shortly presents the Modelica language. Also, in this part we compare the Modelica language and the Web Ontology Language (OWL) [18] developed in the Semantic Web community [5]. Section 3 enters into the details of some parts of the Modelica Standard Library and shows an example of its usage. Section 4 presents conclusions and future work.

## 2 Modelica

Modelica [4, 7, 8, 9] is an object-oriented declarative language used for modeling of large and heterogeneous physical systems. The Modelica language is a new, revolutionizing approach to physical modeling area because is component-based and equation-based, which provide strong reuse (equations are more powerful than assignments because they do not specify control flow). Modelica has a general class concept in which documentation, attributes (components) and the class behavior can be stated. Modelica libraries have detailed formal semantics based on algebraic, differential and difference equations. Modelica language provides constructs for building class documentation (both textual and icons), which can be used by tools to provide visual modeling. Also, in Modelica the


[1] Email: adrpo@ida.liu.se
[2] Email: petfr@ida.liu.se

connections between components are clearly specified with the use of connectors.

For modeling with Modelica, commercial software products such as MathModelica [3] or Dymola [1] have been developed. However, there are also open-source projects like the OpenModelica Project [10]. We briefly introduce the Modelica language by a short example:

```
class HelloWorld
 Real x(start = 1);
 parameter Real a = 1;
equation
 der(x) = -a*x;
end HelloWorld;
```

The example defines a simple class with two attributes and one equation section. This simple model can be configured when used again in other models i.e. `HelloWorld(a=3);`

Comparison of provided functionality between Modelica, Unified Modeling Language (UML) [6] and RosettaNet [17] technical dictionary is discussed in [13]. The conclusion is that sharing and reuse of static engineering ontologies among these languages can be fully automated.

When comparing Modelica and the Web Ontology Language (OWL) [18] developed in the Semantic Web we can outline the following:

- Classes are template-based in Modelica vs. classes are constructed from several primitives using logical connectors in OWL.
- In OWL relations between classes can be specified and additional constraints can be stated. Also reasoner tools provide the possibility of inferring new knowledge from existing facts.
- Both languages have multiple inheritance, subtyping and XML serialization (ModelicaXML [15] for Modelica).

Modelica users and library developers would benefit from Semantic Web technologies and research work is in progress to adapt these to Modelica.

# 3 Modelica Standard Library (MSL)

In this section we shortly introduce the Modelica Standard Library (MSL) and give a usage example. For space reasons we prompt the interested reader to the detailed description of the MSL, available at:
http://www.modelica.org/libraries.shtml

## 3.1 Overview of the ontology

The ontology is structured into several sub-ontologies (packages):

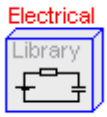
**Modelica.Blocks - Input/Output blocks**


This package provides input/ output blocks for building up block diagrams.

**Modelica.Constants – Mathematical and physical constants**


This package defines often needed constants from mathematics, machine dependent constants and constants from nature.

**Modelica.Electrical – Electric and electronic components**




This package contains electrical components to build up analog circuits.

**Modelica.Icons – Icon definitions of general interest**


This package contains icon definition used to document components (for visual modeling).

**Modelica.Math – Mathematical functions**


This package defines highly used mathematical functions (sin, cos, tan, etc).

**Modelica.Mechanics – Mechanical components (one dimensional rotational and translational)**


This package defines components to model mechanical systems.

**Modelica.Thermal – Thermal components**


This package defines components to model one dimensional heat transfer with lumped elements.

**Modelica.Siunits – SI-unit type definitions (according to ISO 31-1992)**


This package provides predefined types such as *Mass, Length, Time*, etc, based on the international standards on units.

## 3.2 Discussion on the Modelica Standard Library

The features of the Modelica language and Modelica tools have made easy for designers to create models. The Modelica Standard Library provides a shared repository of components for re-use in different models. Tools like MathModelica are using the Modelica Standard Library to help users visually pick and connect components into larger models as in Figure 1.

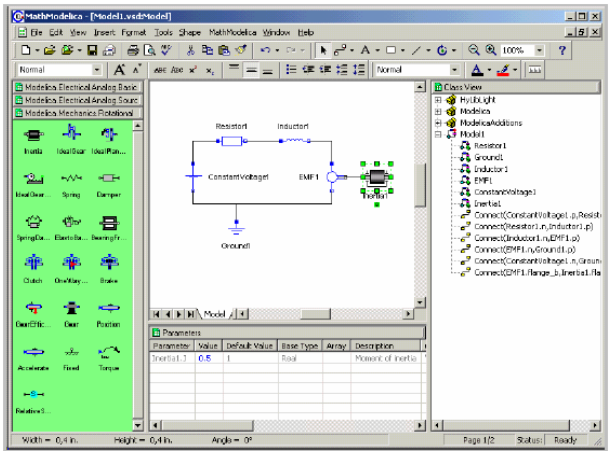


**Figure 1: Visual construction of models.**

From the left part of the Figure 1 the components of a MSL library can be dragged into the current model where they can be connected and further configured. Because the components are very generic and highly configurable they can be easily re-used in different models or different parts of the same model.

The library developers can impose certain week restrictions on the use of the components to ensure that they cannot be misused. However, the Modelica language lacks the power of imposing advanced constraints on the components or their relationship. We will address this issue in the future by translating Modelica to the Web Ontology Language (OWL) [18] and use this language to express restrictions, additional domain knowledge, distributed use of models over the WWW, etc. A short example of translating Modelica to OWL is given in the Appendix.

## 3.3 Example

As an example of Modelica Standard Library (MSL) use, we present the model of a DC-motor. The visual layout of this model is presented in Figure 2. Additional examples can be found at the Modelica website [7, 8].

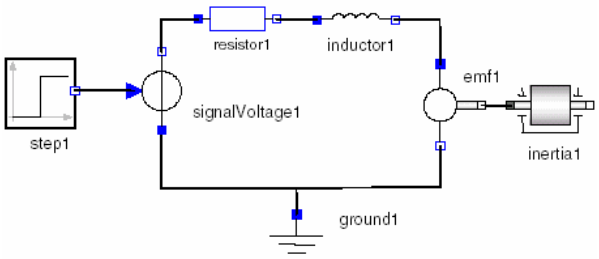



**Figure 2: DC-motor model**

The model presented contains components from three domains that can be found in the `Modelica.Mechanics`, `Modelica.Electrical` and `Modelica.Blocks` sub-libraries of MSL. The code for the DC-motor is as follows:

```
model DCMotor
  import Modelica.Electrical;
  import Modelica.Mechanics;
  import Modelica.Blocks;
  Inductor inductor1;
  Resistor resistor1;
  Ground ground1;
  EMF emf1;
  SignalVoltage signalVoltage1;
  Step step1;
  Inertia inertia1;
equation
  connect(step1.y,
          signalVoltage1.voltage);
  connect(signalVoltage1.n,
          resistor1.p);
  connect(resistor1.n, inductor1.p);
```

```
  connect(signalVoltage1.p, ground1.p);
  connect(ground1.p, emf1.n);
  connect(inductor1.n, emf1.p);
  connect(emf1.rotFlange_b,
          inertia1.rotFlange_a);
end DCMotor;
```

The connections between components are realized by the `connect` statement and can only be established between connectors of equivalent types. This ensures that only valid connections can be made between components.

A model definition can import several packages in order to use the classes defined in them. The packages can be extended through inheritance or specialized through redeclaration. The imports can be named (i.e. `import SI=Modelica.Siunits`), qualified or unqualified (import everything). These features provide a detailed control over the imported definitions and help avoid name conflicts.

In this paper we focused more on the model design part and less on the simulation of the created models. For simulation the models are checked for correctness according to the Modelica static semantics, flattened and translated to highly efficient C code glued with numerical solvers. We simulate the DC-motor model and plot a few of its variables in Figure 3.

```
simulate(DCMotor,stopTime=25);
plot({step1.y,inertia1.flange_a.tau})
```

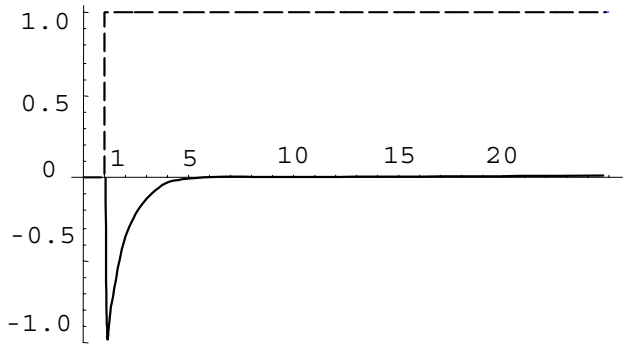



**Figure 4. DCMotorCircuit simulation with plot of input signal voltage step and the flange angle.**

## 4 Conclusions and Future Work

We have presented parts of the Modelica Standard Library (MSL) and showed how MSL is used when building models. We have also outlined the main similarities and differences between Modelica and other ontology languages [18] developed in the Semantic Web [5] community.

As future work we would like to automatically construct an ontology translated from MSL into the Web Ontology Language (OWL). We can foresee that the structural part of the Modelica classes can be translated easily into OWL as we briefly show in the Appendix. The non-trivial part would be to build the relationships between the translated concepts. Such relationships would require additional ontologies that provide concepts for system decomposition, physical processes, etc. These ontologies, combined with the Semantic Web technologies would add new functionality to Modelica tools like:

- Classifying new concepts (classes) and verifying models (i.e. that a model is coherent, etc)
- Imposing additional restriction over the models (i.e. an electric circuit must have a ground component, a car must have 4 wheels, etc)
- Expressing some of the Modelica static semantics directly in OWL (inheritance, subtyping, etc).

## 5 Acknowledgements

We would like to thank to all people in the Modelica community who are actively involved in the development and maintenance of the Modelica Standard Library.

## 6 References

1. Dynasim. *Dymola*, http://www.dynasim.se/.
2. INCOSE. *International Council on System Engineering*, http://www.incose.org.
3. MathCore. *MathModelica*, http://www.mathcore.se/.
4. *Modelica: A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification version 2.1*, Modelica Association, 2003,
5. *Semantic Web Community Portal*, http://www.semanticweb.org/.

6. OMG. *Unified Modeling Language*, http://www.omg.org/uml.
7. Modelica Community, http://www.modelica.org/
8. Moldelica Libraries (Ontologies), http://www.modelica.org/libraries.shtml
9. Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley-IEEE Press, 2003, http://www.mathcore.com/drmodelica.
10. Peter Fritzson, Peter Aronsson, Peter Bunus, Vadim Engelson, Levon Saldamli, Henrik Johansson and Andreas Karström. *The Open Source Modelica Project*, in *Proceedings of The 2th International Modelica Conference*,March 18-19, 2002, Munich, Germany.
11. Adrian Pop, Peter Fritzson. *ModelicaXML: A Modelica XML representation with Applications*, in *International Modelica Conference*,3-4 November, 2003, Linköping, Sweden.
12. Adrian Pop, Ilie Savga, Uwe Assmann and Peter Fritzson. *Composition of XML dialects: A ModelicaXML case study*, in *Software Composition Workshop 2004, affiliated with ETAPS 2004*,3 April, 2004, Barcelona.
13. Olof Johansson, Adrian Pop, Peter Fritzson, *A functionality Coverage Analysis of Industrially used Ontology Languages,* in *Model Driven Architecture: Foundations and Applications (MDAFA), 2004,* 10-11 June, 2004, Linköping, Sweden.
14. Pim Borst, Hans Akkermans and Jan Top, *Engineering ontologies*, in Int. J. Human-Computer Studies, 1997, no. 46, p 365-406
15. Adrian Pop, Olof Johansson and Peter Fritzson, *An integrated framework for model-driven product design and development using Modelica*, accepted for publication at the 45th Conference on Simulation and Modeling (SIMS), 23-24 September 2004, Copenhagen.
16. Mogens Myrup Andreasen. *Machine Design Methods Based on a Systematic Approach (Syntesemetoder pa systemgrundlag)*, Lund Technical University, Lund, Sweden, 1980
17 RosettaNet, http://www.rosettanet.org
18. Word Wide Web Consortium (W3C). Web Ontology Language (OWL), http://www.w3.org/TR/2003/CR-owl-features-20030818/

# 7 Appendix

In this section we show a simple example of how structural parts of Modelica could be translated into OWL. This kind of translation could be further augmented with additional constraints or information. Also, an OWL validator would be able to check such documents.

The following Modelica models and their translation into OWL are presented in the following:

```
class Body            "Generic body"
  Real mass;
  String name;
end Body;
class CelestialBody "Celestial body"
  extends Body;
  constant Real g = 6.672e-11;
  parameter Real radius;
end CelestialBody;

CelestialBody moon(name = "moon",
     mass = 7.382e22, radius =
1.738e6);

Body body_instance(name = "some body",
     mass = 7.382e22);
```

Our Modelica model has two classes (concepts) **Body** and **CelestialBody** the latter being a subclass of the former (by using "extends" statement).

The encoding in OWL is as follows:

```
<?xml version="1.0" ?>
<rdf:RDF

    <!-- namespaces declaration -->
    xmlns=".../inheritance.owl#"
     xmlns:modelica=".../inheritance.owl#"
    xml:base=".../inheritance.owl">

 <owl:Ontology rdf:about=
         ".../inheritance.owl" />

  <!-- define Body -->
  <owl:Class rdf:ID="Body">
    <rdfs:label>Generic Body</rdfs:label>
  </owl:Class>
  <!-- define mass -->
  <owl:DatatypeProperty rdf:ID="mass">
     <rdfs:domain rdf:resource="#Body"/>
     <rdfs:range
          rdf:resource="XMLSchema#float"/>
```

```
</owl:DatatypeProperty>
<!-- define name -->
<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="#Body"/>
  <rdfs:range
     rdf:resource="XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- define CelestialBody -->
<owl:Class rdf:ID="CelestialBody">
  <rdfs:label>
   Celestial Body
  </rdfs:label>
   <rdfs:subClassOf
        rdf:resource="#Body" />
   <!-- cardinality restriction on the
 g constant: one and only one in
 CelestialBody -->
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#g"/>
    <owl:cardinality
      rdf:datatype
      ="XMLSchema#nonNegativeInteger">
         1
    </owl:cardinality>
   </owl:Restriction>
  </rdfs:subClassOf>
 </owl:Class>
 <!-- define g -->
 <owl:DatatypeProperty rdf:ID="g">
   <rdfs:domain
       rdf:resource="#CelestialBody"/>
   <rdfs:range rdf:resource=
      "XMLSchema#float"/>
 </owl:DatatypeProperty>
 <!-- define radius -->
 <owl:DatatypeProperty rdf:ID="radius">
    <rdfs:domain
        rdf:resource="#CelestialBody"/>
   <rdfs:range
        rdf:resource=
         "XMLSchema#float"/>
 </owl:DatatypeProperty>
 <!--
  instance declaration of CelestialBody
 -->
 <CelestialBody rdf:ID="moon">
  <name
    rdf:datatype="XMLSchema#string">
     moon
  </name>
  <mass rdf:datatype="XMLSchema#float">
     7.382e22
  </mass>
  <radius
     rdf:datatype="XMLSchema#float">
     1.738e6
  </radius>
  <g rdf:datatype="XMLSchema#float">
     6.672e-11
  </g>
 </CelestialBody>
 <!-- instance declaration of Body
 -->
 <Body rdf:ID="body_instance">
  <name
    rdf:datatype="XMLSchema#string">
    some body
  </name>
  <mass rdf:datatype="XMLSchema#float">
    7.382e22
  </mass>
 </Body>
</rdf:RDF>
```