Unit Checking and Quantity Conservation

Sven Erik Mattsson Hilding Elmqvist
Dynasim AB
Ideon Science Park, SE 223 70 Lund, Sweden
SvenErik.Mattsson@3ds.com Hilding.Elmqvist@3ds.com

Abstract

What can be done to guaranty correctness of a model? The paper discusses two approaches to automatic checking. First, Dymola's support of units, unit checking and unit deduction is described. It has already proven useful and has helped improving the quality of the Modelica Standard Library. The display unit concept allows users to enter parameters and plot variables in different units. The inputs, outputs and parameters of general blocks defining sources and mathematical operations have of course no units specified. Dymola infers their units in order to improve the variable browsers for entering parameter values and plotting variables during simulation. Second, the possibilities of checking quantity conservation automatically are discussed. It is in open area with a large potential to check that models fulfill the very basic laws of physics including energy conservation, Newton's third law "action equals reaction", etc. To really support automatic checking of quantity conservation it is necessary to include more information in the models. Fortunately, it seems as if most of this can be done in the basic components such as inertia, body, volume, capacitor etc which actually store some quantities and in dissipative elements as for example resistors or friction elements.

1 Introduction

Modelica (Modelica, 2007) is a powerful modeling language. It allows you to quickly build complex models by putting together model components from free public and commercial libraries. The openness of Modelica makes it easy to modify an existing component. All this opens for errors. How can we guide users and provide automatic checking? How can we guarantee quality of provided library components?

Modelica is a strongly typed language implying that classical computer scientific methods can be used. The Modelica 3.0 definition has taken this further and introduced the concepts of plug-in compatibility and balanced models. This paper will discuss two other orthogonal approaches:

1. unit checking of expressions and equations
2. checking of quantity conservation.

In Sections 2-4, Dymola's support of units, unit checking and unit deduction is discussed. In Section 5 the possibilities of checking quantity conservation automatically are discussed.

2 Support of Units in Dymola

Physical modeling deals with physical quantities such as length, mass, force, current. The value of a quantity is generally expressed as the product of a number and a unit. Modelica (2007) supports this approach. A real variable have a quantity attribute and a unit attribute, for example

```
type Mass = Real(quantity="Mass",
                 final unit="kg");
```

The package Modelica.SIunits provides a large set of predefined quantities and it is recommended to use them whenever possible.

2.1 SI units

The Modelica specification states "A basic support of units in Modelica should know the basic and derived units of the SI system." Dymola fulfils this requirement.

A good reference on SI units is what commonly is called the SI brochure published by Bureau International des Poids et Mesures [BIPM, 2006]. The NIST Reference on Constants, Units, and Uncertainty [NIST, 2000] gives a good overview; see also [Taylor, 1995]. ISO does not specify a formal syntax for unit expressions but there are strict recommendations. The Modelica language specification includes a formal specification based on these recommendations.

Dymola supports all the 20 SI prefixes to form decimal multiples and submultiples of SI units.

Factor	Name	Symbol	Factor	Name	Symbol
10^{1}	deca	da	10^{-1}	deci	d
10^{2}	hecto	h	10^{-2}	centi	c
10^{3}	kilo	k	10^{-3}	milli	m
10^{6}	mega	M	10^{-6}	micro	μ
10^{9}	giga	G	10^{-9}	nano	n
10^{12}	tera	T	10^{-12}	pico	p
10^{15}	peta	P	10^{-15}	femto	f
10^{18}	exa	E	10^{-18}	atto	a
10^{21}	zetta	Z	10^{-21}	zepto	z
10^{24}	yotta	Y	10^{-24}	yocto	y

Dymola knows all the seven SI base units

Name	Symbol
metre	m
kilogram	kg
second	s
ampere	A
kelvin	K
mole	mol
candela	cd

as well as the 22 SI derived units that have been given special names and symbols

Name	Symbol (in Modelica)	Definition
radian	rad	1
steradian	sr	1
hertz	Hz	1/s
newton	N	kg.m/s^2
pascal	Pa	N/m^2
joule	J	N.m
watt	W	J/s
coloumb	C	A.s
volt	V	W/A
farad	F	C/V
ohm	Ohm	V/A
siemens	S	A/V
weber	Wb	V.s
tesla	T	Wb/m^2
henry	H	Wb/A
degree Celcius	degC	K
lumen	lm	cd.sr
lux	lx	lm/m^2
becquerel	Bq	1/s
gray	Gy	J/kg
sievert	Sv	J/kg
katal	kat	mol/s

There are also units that are not part of the International System of Units, that is, they are outside the SI, but they are accepted for use with the SI. Dymola knows the following of them:

Name	Symbol	Expressed in SI units
minute	min	60 s
hour	h	60 min
day	d	24 h
degree	deg	(π/180) rad
litre	l	dm^3
decibel	dB	1
electronvolt	eV	0.160218 aJ
bar	bar	0.1 MPa
phon	phon	1
sone	sone	1

In power systems the unit for apparent power is “V.A”. Dymola knows var = V.A which has been adopted by the International Electrotechnical Commission, IEC, as the coherent SI unit volt ampere for reactive power, see IEC [2007].

The rotational frequency *n* of a rotating body is defined to be the number of revolutions it makes in a time interval divided by that time interval. The SI unit of this quantity is thus the reciprocal second, s^{-1}. However, the designations "revolutions per second" (r/s) and "revolutions per minute" (r/min) are widely used as units for rotational frequency in specifications on rotating machinery. Although use of rpm as an abbreviation is common, its use as a symbol is discouraged. Dymola knows r = 2π rad. It can be used for example as “r/s” or “r/min”.

Dymola also knows the temperature units degF (degree Fahrenheit) and degRk (degree Rankin).

2.2 Other units

Dymola recognizes the users' needs to enter parameters and plot variables in different units. Modelica

defines displayUnit for that purpose. Dymola supports displayUnit when plotting variables and when entering values in parameter dialogs.

A user can define units for display and its meaning is in terms of the SI unit. For example, the display unit "min" is defined in the following way in terms of the SI unit "s" as

```
defineUnitConversion("s", "min", 1/60);
```

There is a fourth optional argument to specify offset. For example, conversion from Kelvin to degrees Fahrenheit can be specified as

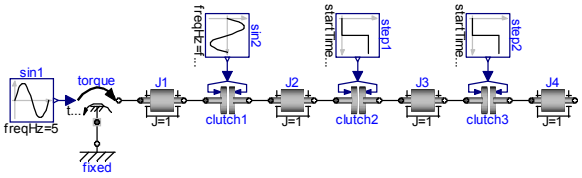
```
defineUnitConversion("K", "degF",
        9.0/5.0, 32-(9.0/5.0)*273.15);
```

However, if the quantity represents a temperature difference the offset shall not be included. Dymola supports an annotation `__Dymola_absoluteValue` to control this. In Modelica.SIunits the quantity temperature difference is specified as

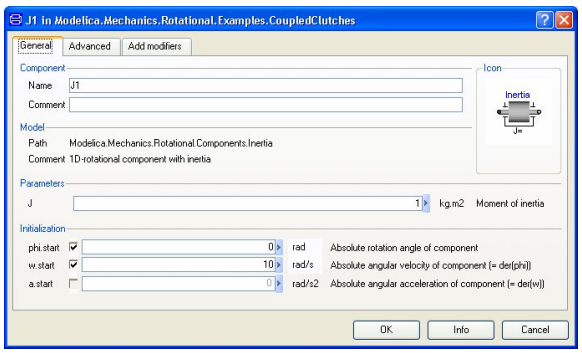
```
type TemperatureDifference = Real (
  final quantity=
          "ThermodynamicTemperature",
  final unit="K")
  annotation
       (__Dymola_absoluteValue=false);
```

These definitions are conveniently stored in script (mos) file that is executed at the start of Dymola. By default Dymola has a file displayUnit.mos including display units of general interest.

As an example, consider the model CoupledClutches in Modelica Standard Library 3.0.



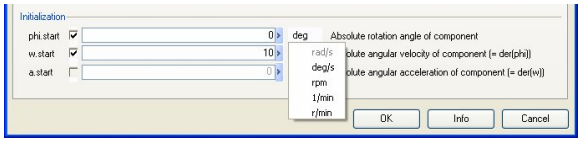
Pop the parameter dialog for the rotating body, J1.



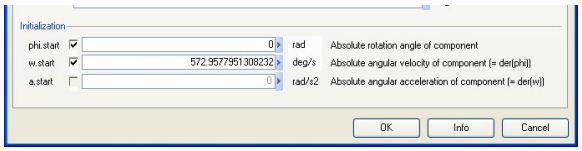
The start values for the angle, phi, and the angular velocity, w, can be entered in SI units. The Modelica code for J1 is

```
Modelica.Mechanics.Rotational.Inertia
  J1(J=1,
     phi(fixed=true, start=0),
     w(start=10, fixed=true))
```

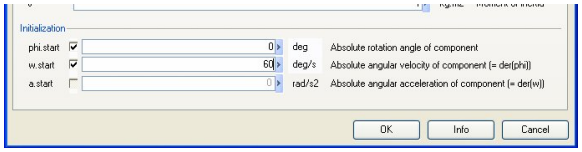
We can enter the start velocity in "deg/s". Click on the unit to pop a menu and select unit



Which alternatives that are available depends on which defineUnitConversion calls that actually have been invoked. It can be customized by any user by editing the file displayUnit.mos. If a user wants to see any length only in mm or inch, the user can restrict the display unit to that.



The value is now displayed as 572.96 deg/s. It is easy to enter a new value, say 60 deg/s.

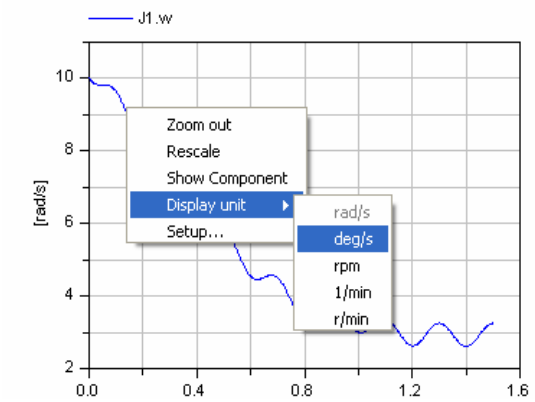


The Modelica code for J1 becomes

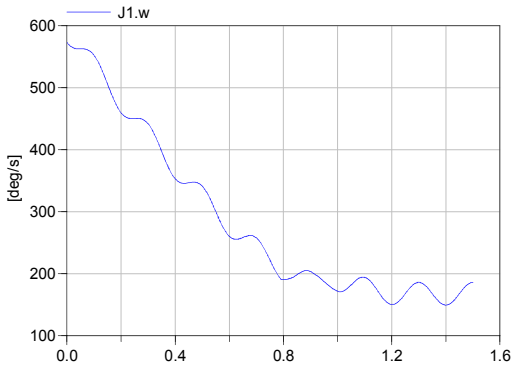
```
Modelica.Mechanics.Rotational.Inertia
 J1(J=1,
    phi(fixed=true, start=0),
    w(start=1.047197551196598,
      fixed=true, displayUnit="deg/s"))
```

Note, that the attribute displayUnit is modified according to our choice. However, the parameter value being 60 deg/s is stored in SI units, "rad/s". Thus portability is preserved and it is still a tool issue to support the displayUnit in the dialogs.

Let us simulate the original example and plot J1.w. Put the cursor on the curve and pop the context menu.



Selecting "deg/s" as unit for plotting gives the plot.



3 Unit Checking

Equations add terms. Naturally these must be of the same physical quantity. This is exploited in the classical physical dimension check of equations which many of you have done by paper and pen in school. Dymola (Dynasim, 2007) has automated this check.

The number of physical quantities we can think of is large. Fortunately, they are related and all physical quantities can be expressed as product of powers of a small set of base quantities. The International System of Units, the SI system, defines such a set including seven physical quantities: length, mass, time, electric current, thermodynamic temperature, amount of substance and luminous intensity, see BIPM (2006). The SI base units define a unit for each of these seven quantities. The units for other quantities are derived. For example, the unit for area is m^2 because the physical quantity area = length*length and the unit for length is m (meter). Thus there is a mapping from quantity to unit in terms of the seven SI base units. Dymola exploits this for unit checking.

Dymola's checking of units is active when checking a package, function or model as well as when translating a model for simulation. It includes checking of unit strings and unit compatibility of equations. It can be seen as a part of the type checking. It includes the checking of actual function input arguments and output arguments against their formal declarations.

Currently Dymola makes a relaxed checking. It means that an empty unit string, "", is interpreted as unknown unit. Also number literals are interpreted to have unknown unit. The unknown unit is propagated according to simple rules

unknown unit * "unit1" -> unknown unit

unknown unit + "unit1" -> "unit1"

There is one important exception. Let e be a scalar real expression. Consider the inverse of e given as 1/e. The number 1 (one) in the numerator does not relax the checking. If e has a well-defined unit then also 1/e has a well-defined unit.

The unit checking is applied to the original equations. This has implications for vector, matrix and array equations. For an array where all elements have the same unit, the check works as if it was a scalar. Arrays and array expressions where the elements have different units are allowed. However, the check is then relaxed and the array is viewed to have an unknown unit that is compatible with all units. Checking the unit consistency between two records is done recursively for each component.

Currently, the unit checking does not issue error messages but it generates only warnings. The unit checking can be disabled.

As a simple example consider the modeling of motion where there is a mistake

```
  parameter Modelica.SIunits.Mass m=1;
  Modelica.SIunits.Velocity v;
  Modelica.SIunits.Force    f;
equation
  m*v = f; //Should read  m*der(v) = f;
```

When checking or translating it, Dymola outputs

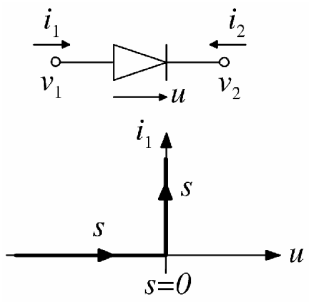
```
Warning: Incompatible units in
  m*v = f;
The part
  m*v
has unit N.s
The part
  f
has unit N
```

Dymola's unit checking has already been proven useful. Several errors in the Modelica Standard Library were found. A user reported that he several years ago had rewritten a model in Modelica, but he did not get the same simulation result. He had really

tried to find the reason, without success. Dymola's unit checking pointed out an inconsistency and he had found the error.

The basic laws for conservation of mass, momentum, electrical charge, energy are expressed as balance equations between physical quantities. Also constitutive equations such as Ohm's law are readily expressed as equations for physical quantities. It means that the unit checking should not make the modeling more complicated in most cases.

However, the parameterized curve descriptions used to model idealized characteristics of for example diodes or Coulomb friction needs more attentions. Consider the modeling of an ideal diode having the characteristics shown in the figure.



The parameterized curve description is

```
off = s < 0;
v = if off then s else 0;
i = if off then 0 else s;
```

The curve parameter is just a real variable that is either representing a voltage or a current. To make the equations unit consistent, the equations can for example be rewritten as

```
v = unitVoltage*(if off then s else 0);
i = unitCurrent*(if off then 0 else s);
```

The s parameter and the unit constants are declared protected as

```
protected
  Real s(final unit="1");
  constant Modelica.SIunits.Voltage
    unitVoltage= 1
     annotation(HideResult=true);
  constant Modelica.SIunits.Current
    unitCurrent= 1
      annotation(HideResult=true);
```

The HideResult annotation has the effect that the unit constants are not included in the simulation result. Basically these constant are only active during unit checking and then eliminated in the equations.

In summary, don't just declare real variables, but declare physical quantities. Use the predefined quantities available in Modelica.SIunits whenever possible. The SI units were invented to allow equations to be written in a clean way without conversion factors. This simplicity is a very good reason for using the SI units in physical modeling. Thus, it is recommended that unscaled SI units are used when specifying the unit attribute of a real variable. To be clear, this also means that prefixes shall not be used. For example "m", "kg", "V", "N.m" and "W" are good, but not "cm", "g", "kV", "MW" or "bar". The displayUnit concept provides convenient entering of parameter values and displaying and plotting of results in other units.

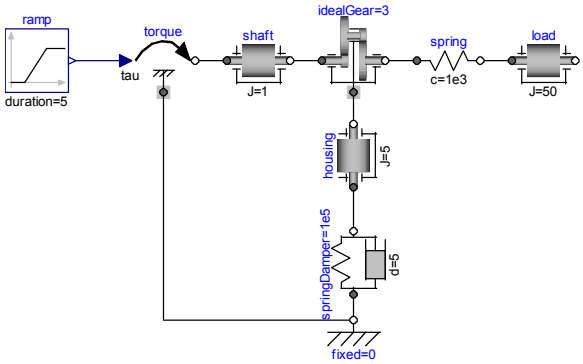
4 Unit Deduction

The Modelica.Blocks library includes general blocks to define sources and mathematical operations. Their inputs and output have of course no units specified. For user convenience, Dymola has introduced automatic deduction of units. Here is a short description Consider the expression, e1 + e2, where Dymola has found that the expression e1 has a well-defined unit u1, but the unit of the expression e2 is unknown. We can then deduce as described in the introduction that the unit of the sum e1 + e2 is u1. Moreover, for unit consistency reasons the unit of e2 must also be u1. If now e2 is a simple variable reference, v, we can deduce that v must have the unit u1. For more complex expressions Dymola makes a downwards recursion to see if it is possible to deduce units of variables with unknown units.

The SignalType definition in the Modelica Standard Library 2.0 allowed the user to specify the units manually by declaring the type of the inputs, the outputs and the parameters of the block. The SignalType is removed in the Modelica Standard Library 3.0 and the units are deduced automatically.

The deduction of units may reveal unit inconsistencies. In such a case it may be useful to enable the logging and inspect the log. It is also useful to check the log when developing a model component, because if a real variable gets its unit deduced that may indicate that the variables shall be declared using any of the quantities defined by Modelica.SIunits.

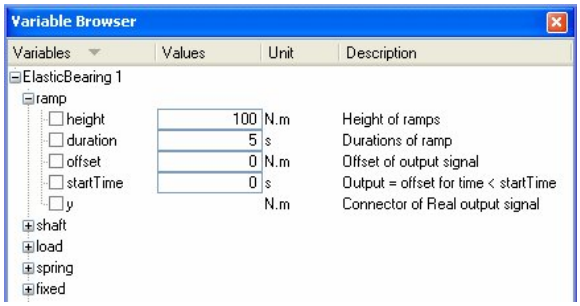
As an example consider the model ElasticBearing in Modelica.Mechanics.Rotational.Examples.



The component ramp is of the class Modelica.Blocks.Sources.Ramp. The parameters

```
parameter Real height=1
                    "Height of ramps";
parameter Real offset=0
               "Offset of output signal";
```

have no units specified. Similarly the unit of the output y is not specified. At translation Dymola deduces their units and displays them in the variable browser



The deduced SI unit radian is treated in a special way by Dymola. Consider Euler's equation for a one dimensional rotating body

```
J*a = flange_a.tau + flange_b.tau;
```

where the inertia, J, has unit kg.m2, the rotational acceleration, a, has unit rad/s2 and the torques flange_a.tau and flange_b.tau have the unit N.m. It means that the left hand side of the equation has the unit, rad.kg.m2/s2 and the right hand side has the unit, N.m = kg.m2/s2. It means that the units are equal besides the left side has a factor "rad". This is fine from the formal point of view because the derived unit radian is formally expressed as m/m, see Table 3 in [BIPM, 2006], which also states that the radian is "a special name for the number one that may be used to convey information about the quantity concerned."

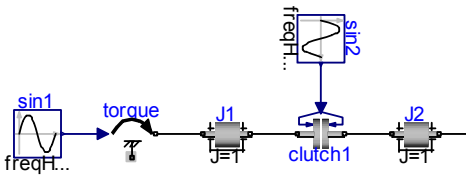
However, in order to support the use of radians when deducing units, Dymola treats the radians as if it was a SI base unit during the analysis. The consistency checking is of course relaxed for radians. The resulting unit will include the minimum power of radians.

5 Quantity conservation

The design of Modelica.Mechanics.Rotational and the discussion on the modeling of mounting have clearly indicated the need for more automatic testing of models. The failure to model the mounting of a drive-train element is an example where the user fails to account for important interactions between components or between a component and its environment. The failure to model the mounting of a component gives a simulation result where momentum is not preserved. Such a model violates Newton's third law "action equals reaction".

Balance equations and conservation of physical quantities such as mass, momentum, energy and electrical charge are basic in physical modeling.

A flow variable of a connector represents the flow of a conserved quantity into a component. Thus it is straightforward to calculate the net amount of a conserved quantity flowing into a component. A prototype implementation has been made in Dymola. As an example consider the model



The flow variables of the connectors have their attributes quantity="Torque". Dymola introduces for each component a variable named sum_Torque and an equation such as

J2.sum_Torque = J2.flange_a.tau+J2.flange_b.tau;

For the clutch that has no inertia, the sum_Torque variables are zero as it should be. It is not zero for inertia models J1 and J2 because they can store momentum. Their models include the equation

J*a = flange_a.tau + flange_b.tau;

Thus, for inertia we have

sum_Torque = J*a

The variable torque.sum_Torque is non zero. For the system, above momentum is not preserved. In reality the drive train is mounted in the car. The chassis provides a corresponding reaction torque, which propagates through the wheels and tires to the road. Thus if we put the drive train above into a chassis model without connecting the bearing connectors of the drive train properly to model the real mounting, we will get wrong simulation results. How can we provide some automatic checks?

For components not storing conserved quantities, we can use sum_Torque and the other sum variables and add an assertion that the sum should be zero. For the example, the simulation stops issuing

Assertion failed: abs(torque.sum_Torque)< 1E-005
Torque not conserved in the component torque.

In the general case, a component needs to include information on what are the storage terms. Tiller and Kittirungsi (2006) propose annotation to be used. For example to indicate that the term J*a above implies storage:

```
Modelica.SIunits.Torque
   torqueStorage = J*a
     annotation(storageTerm));
```

In 3 D mechanics forces are vectors and the balances of forces must be set up in the same frame. Moreover, the torque balances are even more complicated since they also include terms referring to the forces acting on the body.

One idea is to add an annotation to Modelica.Mechanics.MultiBody.Interfaces.Frame:

```
connector Frame
  "Frame of a mechanical system"
  annotation(ConservedQuantity(
    Force= Frames.resolve1(R.T, f),
    Torque=Frames.resolve1(R.T,t)+
     cross(r,Frames.resolve1(R.T,f))));

 import SI = Modelica.SIunits;
 SI.Position r_0[3];
 Frames.Orientation R;
 flow SI.Force f[3];
 flow SI.Torque t[3];
end Frame;
```

The scope for the right-hand-sides is the local connect (exactly as for e.g. a binding equation in the class) and this takes precedence over the default-summing of quantity-flows to 0. This annotation should only be added once [i.e. for the base-class of all flange-connector and not for each model], and we could alternatively have this built-in in Dymola for this class.

Please, note that if a model component fails to annotate or mark a term as contribution to storage then the check will detect this, i.e., the component model is not conserving properly.

In order not to be forced to model all universe, it is necessary to support infinite sources or sinks for conserved quantities. Again it is possible to use an annotation to mark such components. However, there is a potential risk with ground elements. Assume that we fix the coupled clutch model above by connecting a ground component to the bearing connector of the component torque. This component can be viewed as a rig where we put the drive train for testing. The rig may be viewed as representing the "infinite mass" of the earth.

Energy conservation is important to check. However, it is more complex. For thermodynamics the heat flows as well as the enthalpy flows are a power flow. However, the energy flow does not always appear explicitly as flow variables in the connectors. Some energy flows can be computed by multiplying the flow quantities by a proper derivative of the corresponding across variable. Examples:

pin.v*pin.i	=>	V*A=W
der(flange.s)*flange.f	=>	m/s*N=W
der(flange.phi)*flange.tau	=>	rad/s*Nm=W

It works for Electrical, Rotational and Translational. For MultiBody there is the problem with different coordinate systems.

Establishing energy conservation also includes identification of energy dissipation. For example a resistor "dissipates" energy, or more explicitly, it converts electrical energy into heat. The basic components in Electrical do not include such information.

The automatic checking of conservation may have several objectives. A primary objective is to catch model errors. However, a conservation condition may be violated over time due to numerical drift of the numerical solution for the conserved quantity. This calls for a more sophisticated checking considering the numerical drift. On the other hand it may also be used to improve the numerical solution. A numerical solver may exploit these invariants for automatic selection of tolerances, i.e. the user put tolerances on invariants. Projection methods may be used to numerically control the drift.

Evidently there is a need to include more information in the models in order to be able to perform automatic checking of quantity conservation. Fortunately, quantity storing is done in the basic components such as inertia, body, volume. For energy balances we need also to consider dissipation in for example resistor, damper, pipe friction etc.

6 Conclusions

Dymola's support of units, unit checking and unit deduction has been described. It has already proven useful. Several errors in the Modelica Standard Library were found. It has encouraged the developers of the Modelica Standard Library to declare vari-

ables representing quantities appropriately. The displayUnit concept allows users to enter parameters and plot variables in different units while allowing clean equations without complicating conversion factors because the equations can refer to the quantities in SI units. The Modelica.Blocks library includes general blocks to define sources and mathematical operations. Their inputs and outputs have of course no units specified. This may also be the case for some parameters such as gain. At translation of a model for simulation Dymola infers their units in order to improve the variable browsers for entering parameter values and plotting variables during simulation.

Second, the possibilities of checking quantity conservation automatically are discussed. It is in open area where there is a large potential to check that models fulfill the very basic laws of physics including energy conservation, Newton's third law "action equals reaction". Evidently there is a need to include more information in the models to really support automatic checking of quantity conservation and there is a need for extensions of Modelica. Fortunately, it seems as if most of this can be done in the basic components such as inertia, body, volume, capacitor etc which actually stores some quantity. For energy balances it is also necessary to identify and mark dissipation in resistors, friction elements etc.

References

BIPM 2006. The International System of Units (SI), Bureau International des Poids et Mesures, 8th edition, 2006. Available at www.bipm.org/en/si/si_brochure

Dynasim. 2007. Dymola Version 6.1. Dynasim AB, Lund. Sweden. http://www.dynasim.se/.

Modelica 2007. Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling – Language Specification, Version 3.0, Available in electronic form at www.modelica.org/documents/ModelicaSpec30.pdf

M.Tiller and B. Kittirungsi: UnitTesting. 2006. A library for Modelica unit testing. Proceedings of the 5th Modelica Conference, Vienna, Austria, 2006, Vol. 2, pp. 695-704.