Natural Unit Representation in Modelica

Kevin L. Davies   Christiann J.J. Paredis
Georgia Institute of Technology
Atlanta, Georgia USA

Abstract

A method is presented by which alternative systems of physical units may be represented and utilized in Modelica. The method may be useful in simulating models of physical systems where the base units of the International System of Units (Système international d'unités, SI)—the standard unit system in Modelica—are poorly scaled. It also provides a convenient means to express the values of physical quantities in fields of science and engineering where data is typically represented in other systems of units or where the rank of the system of units is less than that of SI (i.e., natural units). By explicitly expressing the value of a physical quantity as the product of a number and a unit (where the unit is an algebraic variable), the method uses variables that are unit-neutral. Unfortunately, workarounds are necessary in order to implement the method in the current version of the Modelica language. Nonetheless, it may be useful in special applications, and the related discussion may provide valuable insight. In particular, it is shown that there is an apparent conflict in the interpretation of "number" and "value" between Modelica and the International Bureau of Weights and Measures (Bureau International des Poids et Mesures, BIPM).

Keywords: natural units; physical quantities; Modelica; SI

1 Introduction

In the mathematical representation of physical systems, the values of quantities are interrelated through equations that express the behavior of the system over time and space. As stated by the BIPM [5, p. 103]:

> "The value of a quantity is generally expressed as the product of a number and a unit. The unit is simply a particular example of the quantity concerned which is used as a reference, and the number is the ratio of the value of the quantity to the unit."

In general, a unit may be the product of powers of other units, whether they are base units or units derived from the base units in the same manner.

In the Modelica language, physical quantities are typically expressed as instances of the `Real` type [12, p. 46]. The `value` attribute of the instance is the number associated with the value of the quantity (not the value of the quantity, as will be seen). The `unit` attribute is a string that describes the unit by which the value of the quantity is divided to arrive at the number.[i] The `displayUnit` attribute (also a string) describes the unit by which the value should be divided to arrive at the number as it is entered by or presented to the user. Based on the information provided by the `unit` and `displayUnit` attributes, simulation tools may perform unit checking and conversion. The `Real` type contains other attributes as well, including `quantity`, which is another string [8, p. 375].

The `SIunits` subpackage of the Modelica Standard Library contains types that inherent from the `Real` type. The type definitions appropriately modify the `unit`, `displayUnit`, and `quantity` attributes (among others) to represent various physical quantities. The `unit` and `displayUnit` attributes are based on the SI. The `quantity` string is generally used to describe the name of the physical quantity. For example, the `Velocity` type has a `unit` of `"m/s"` and a `quantity` of `"Velocity"`.

If an instance of the `Velocity` type has a `value` of one ($v = 1$), then it is meant that "the value of velocity is equal to one meter per second." Again, the `value` attribute represents the number, or the value divided by the unit, not the value itself. This apparent conflict could be solved in one of several ways. First, the unit could be strictly set equal to be one (1), regardless of what the unit is. This is the essence of the current implementation in Modelica. It is also the interpretation we use when we are working a problem by hand

[i]Hereafter, the value of the quantity is referred to as simply the value, but it should not be confused with the `value` attribute (which, in the current version of the Modelica language, is the number).

and drop the units because we are exclusively using a particular system of units. However, in this case, the statement that "the value of a quantity is generally expressed as the product of a number and a unit" [5] loses its meaning; it may as well be "the value of a quantity is generally expressed as the number." Second, the `value` attribute could be renamed as the `number` attribute. Since the name of a variable is an implicit reference to this attribute (whatever it is called), the variable would then represent the number. The third method of resolution is to let the units (the meter and the second in this case) be mathematical entities and let $v' = 1 \cdot \mathrm{m}/\mathrm{s}$. Here, the variable v' directly represents the value. Its `value` attribute is the value in the context of the statement by the BIPM.

2 Method

The approach is to follow the third method to resolve the apparent misnomer of the `value` attribute—to factor the units out of the `unit` attribute and into the `value` attribute. This offers the advantage that unit conversion is handled naturally. The essence of unit conversion is that the phrase "x (value) in u (unit)" is interpreted mathematically as "x divided by u." Continuing with the previous example, v' is divided by m/s in order to display v' in meters per second (as a number). The result is simply one (1). If the unit foot is established through the appropriate relation ($\mathrm{ft} \approx 0.3048 \cdot \mathrm{m}$) and v' is divided by ft/s, the result is v' in feet per second (~ 3.2894).

As another example, systems involving angle are sometimes evaluated by working with variables in cycles and other times with variables in radians. If the variable is the value, then "variable in unit" means "value divided by unit." If we work with the value directly, then there is no need to specify which unit we are working "in." The unit is included; it has not been factored out by division. As long as the dimensionality is correct, the math is equivalent due to the relationships among units (or combinations of units). In this case, the relevant unit relation is $1 \cdot \mathrm{cycle} = 2\pi \cdot \mathrm{rad}$.[ii] This example extends directly to frequency (angle per time). Often, different symbols are used for frequency in Hz (ν) and frequency in rad/s (ω). If the units are included in the variable f, then $f = \nu \cdot \mathrm{Hz} = \omega \cdot \mathrm{rad}/\mathrm{s}$.

In this method, each unit must be represented by an algebraic variable (albeit constant). For each unit introduced, it is necessary to add an equation that allows the unit's value to be determined. If a unit is considered to be a derived unit, then the equation simply relates the unit to other units (e.g., $1 \cdot \mathrm{cycle} = 2\pi \cdot \mathrm{rad}$). However, there are several units (in SI, 7) that may not be simply defined via other units. These base units must be related to something outside of the algebraic system of equations representing the immediate physical system. This something is the "particular example of the quantity concerned which is used as a reference" quoted previously [5]. The designation of "base" or "derived" is somewhat arbitrary [8, p. 375], but regardless, there are a number of units that must be defined by example. Considering only the immediate physical system, these units are linearly independent.

If only the SI will be used, then it is easiest to strictly set each of the base units of SI equal to one (1)—the meter (m), kilogram (kg), second (s), ampere (A), kelvin (K), mole (mol), and candela (cd). This is implicitly the case in `Modelica.SIunits`, but again, it hardly captures the idea that a value is the product of a number and a unit.

There are systems where typical values are many orders of magnitude larger or smaller than the related product of powers of base SI units (e.g., the domains of astrophysics and atomic physics). In modeling and simulating those systems, it may be advantageous to choose appropriately small or large values (respectively) for the corresponding base units such that the product of the number (large or small in magnitude) and the unit (small or large, respectively) is well-scaled. Products of this type are often involved in initial conditions or parameter expressions, which are not time-varying. Therefore, the number and the unit can be multiplied before the dynamic simulation. During the simulation, only the value is important. After the simulation, the trajectory of the value may be divided by the unit for display. This scaling is usually unnecessary due to the wide range and appropriate distribution of the real numbers that are representable in floating point space. The Modelica language specification recommends that floating point numbers be represented in at least IEEE double precision, which covers magnitudes from $\sim 2.225 \times 10^{-308}$ to $\sim 1.798 \times 10^{308}$ [12, p. 13]. However, in some cases it may be preferable to carefully scale the units and use single precision instead for the sake of computational performance. There are fields of research where, even today, simulations are sometimes performed in single precision [10] and where scaling is a concern [14, p. 29].

[ii] Furthermore, a cycle is typically equated to the number one (1). For instance, in SI, a frequency of one hertz ($1 \cdot \mathrm{Hz}$) is equated to one per second ($1/\mathrm{s}$) [5] even though to be precise it is one cycle per second ($1 \cdot \mathrm{cycle}/\mathrm{s}$).

Since there are many systems of units besides the SI, it is best if the method is neutral with regards to not only the values of the base units, but also the choice of the base units and even the number of base units. As mentioned previously, the choice of base units is somewhat arbitrary, and different systems of units are based on different choices. Some systems of units have fewer base units (lower rank) than SI, since additional constraints are added that exchange base units for derived units. For example, the Planck, Stoney, Hartree, and Ryberg systems of units define the Boltzmann constant to be equal to one ($k = 1$) [15]. The unit K is "eliminated" [9, p. 386] or, more precisely, considered a derived unit instead of a base unit. In the SI, the Boltzmann constant would be derived from the base units kilogram, meter, and second ($\mathrm{K} \approx 1.381 \times 10^{-23} \cdot \mathrm{kg} \cdot \mathrm{m}^2/\mathrm{s}^2$). In such a system, terms that would otherwise be written as kT may be replaced by simply T; temperature (T) is considered to be energy per particle or degree of freedom. In this case, it is not possible to arbitrarily choose a value for K.

A unit is considered to be a "natural" unit if it depends only on values of universal physical constants [15]. If a system of units is purely natural, then all its base "units" are base "constants." The "particular example of the quantity concerned which is used as a reference" [5] is an experiment that yields precise and repeatable results in determining a constant rather than a prototype which is carefully controlled and distributed via replicas. For instance, a natural unit for electrical resistance is the von Klitzing constant, and it can be chosen as a base constant. Often, the base constants are defined to be equal to one. However, just as it is not necessary to set base units to one, it is not necessary to set base constants to one. The values can be chosen to best scale the numerics of the system.

It is judicious to check that the terms of each equation have the same dimension. Fortunately, methods for unit checking have already been established and implemented in Dymola [11]. In the present context, those methods can, in theory, be applied to the dimension instead (i.e., "dimension checking" instead of "unit checking"). Again, in the present method, the unit is included in the `value` attribute. The question of which unit the variable is "in" is not applicable, but it is still possible and appropriate to check the dimensions.

The dimension of a value may be expressed in the same manner as the unit is in the current version of the Modelica language [12, Ch. 18]. For SI, it would be appropriate to use these base dimensions instead of the corresponding base units: length (L), mass (M), time (T), electric current (I), thermodynamic temperature (Theta), amount of substance (N), and luminous intensity (J) [5, p. 105]. In the example that follows, the Rydberg constant, Faraday constant, and the specific mass of electrons are all set equal to one. Therefore, the rank is reduced from seven to four.

3 Implementation

The method is implemented in version 3.2 of the Modelica language [12] and version 7.4 of Dymola [7]. However, the implementation includes several less-than-ideal workarounds; a full and consistent implementation would require changes to the language and the modeling environment (see Sec. 4).

First, it is necessary to define the units and constants as variables. These variables must be declared in an accessible package so that they can be used in equations within the declaration, initial, and dynamic sections of the model and its subclasses. An excerpt from this `Units` package is shown in Listing 1. The top section of the code establishes mathematical constants (in this case, only π). The next section establishes the base constants and units, which are adjustable. The third section establishes the constants and units which may be derived from the base units and constants using accepted empirical relations. The rest of the code (not listed) establishes the SI prefixes and the remaining derived units and constants. The SI prefixes are included in their unabbreviated form in order to avoid name conflicts (e.g., `constant Real kilo(unit="1")=1E3`). In a model, a kilometer is included as `kilo*m`, unless `km` is defined as a stand-alone unit. All of the primary units of SI are included (Tables 1 and 3 of [5]) except for °C, since it involves an offset. Other convenient units are included for the system at hand (e.g., `atm`). For convenience, the `Units` package is given the abbreviated label `U` by an `import` statement at the top level of the entire library or containing package.

Each unit or constant is a `constant Real`. The `unit` attribute is given a string that describes the dimension. The abbreviations l, N, T, and I are used for length, number, time, and luminous intensity, respectively.[iii] The dimensions are combined as strings

[iii]Lowercase "ell" is used so that Dymola 7.4 recognizes it as a unit—the liter. Dymola also recognizes N as newton and T as tesla. This is not the meaning here, but there is no problem since it happens that these three units are orthogonal. As long as lu-

according to the rules established for unit strings in the Modelica language [12, p. 210].

The units, constants, and prefixes must be identically defined in Dymola's workspace so that they can be used to convert values to numbers for display. The definitions from the `Units` package are copied to a Modelica script. All the specifications of `constant Real` and of the `unit` attribute are removed. It is important that the base units or constants are declared at the beginning of the script and all derived units are arranged in an order that allows the script to succeed on the first pass. The script is run when Dymola is launched. Assert statements are added at the end of the script to perform basic checks on the relationships among the values.

Now, types must be defined for the required quantities. Each quantity inherits from the `Real` type. The `unit` attribute is given a string that describes the dimension (as in the `Units` package). The `quantity` attribute is not used, since the type *is* the quantity. The `displayUnit` attribute is given a string that describes the desired unit to be used for display (according to the format specified in Ch. 18 of [12]). By default, it is the simplest expression of the unit in SI. For convenience, the package containing the quantities is given the global, abbreviated label `Q`.

Another Modelica script is written to define the unit conversions for display using Dymola's `defineUnitConversion` command. As mentioned previously, a value is divided by a unit to arrive at a number for display. This script is executed after the script that defines the units, constants, and prefixes (automatically—upon starting Dymola) so that all of those variables are available. For example, the entry for velocity is `defineUnitConversion("1/T", "m/s", s/m)`.

A top-level "environment" model is included which stores copies of the base units or constants. With that information, it is possible to re-derive all of the other units and constants. This is important in order to properly interpret simulation results even after the base units or constants are re-adjusted.

Where the `der` operator is used, it is explicitly divided by the unit second (e.g., `der(x)/U.s`). This is necessary because the global variable `time` is time in seconds.

Listing 1: Selected constants from the `Units` package

```
// -----------------------------------------------
// Base physical constants and units

replaceable constant Bases.Default base
    constrainedby Bases.Basis
"Scaleable base constants and units";
// Note: The base constants and units may be
// replaced to suit the scale of the physical
// system.

final constant Q.Angle rad=base.rad "radian";
final constant Q.Wavenumber R_inf=base.R_inf
"Rydberg constant (R_&infin;)";
final constant Q.Velocity c=base.c
"speed of light in vacuum (c)";
final constant Q.MagneticFluxReciprocal k_J=
    base.k_J
"Josephson constant (k_J)";
final constant Q.Resistance R_K=base.R_K
"von Klitzing constant (R_K)";
final constant Q.RadiantIntensity 'cd'=base.'cd' "
    candela";
final constant Q.Number k_F=base.k_F
"Faraday constant (k_F)";
final constant Q.Number R=base.R "gas constant";

// -----------------------------------------------
// Empirical constants and units
// Note: The values are currently based on the
// those from NIST (2010). The measured (rather
// than conventional) values are used.

constant Q.Length m=10973731.568539*rad/R_inf "
    meter";
// SI unit of length
// This is the "Rydberg constant" relation (NIST,
// 2010). The unit radian is included to be
// explicit, although it is currently one by
// definition (BIPM, 2006).
// (http://en.wikipedia.org/wiki/Rydberg_constant)
    .
constant Q.Time s=299792458*m/c "second";
// SI unit of time or duration
// This is the "speed of light in vacuum" relation
// (NIST, 2010).
constant Q.MagneticFlux Wb=483597.870E9/k_J "weber
    ";
// SI unit of magnetic flux
// This is the "Josephson constant" relation
// (NIST, 2010).
constant Q.Conductance S=25812.8074434/R_K "siemen
    ";
// SI unit of electrical conductance
// This is the "von Klitzing constant" relation
// (NIST, 2010). The unit radian is included on
// the denominator for dimensional consistency,
// but it is one by the current defition (BIPM,
// 2006).
constant Q.ParticleNumber mol=96485.3365*Wb*S/k_F
    "mole";
// SI unit of amount of substance
// This is the "Faraday constant" relation (NIST,
// 2010). The factor Wb*S is the coulomb, which
// is defined below.
```

minous intensity is not represented in the model (I, which is not recognized), unit checking may be used as dimension checking.

```
constant Q.Potential K=8.3144621*(Wb*rad)^2*S/(s*
    mol*R) "kelvin";
// This is the "molar gas constant" relation
// (NIST, 2010). The factor (Wb*rad)^2*S/s is the
// joule, which is defined below.
```

Listing 2: Selected records from the `Units.Bases` package

```
record Basis "Base constants and units"

  final constant Q.Angle rad=1 "radian";
  // SI unit of rotation or planar angle
  constant Q.Wavenumber R_inf=1
    "Rydberg constant (R_&infin;)";
  // The SI unit length (meter) is inversely
  // proportional to this value, which should be
  // increased for larger characteristic lengths.
  constant Q.Velocity c=1 "speed of light in
      vacuum (c)";
  // The SI unit time (second) is inversely
  // proportional to this value (and R_inf), which
  // should be increased for larger characteristic
  // times.
  constant Q.MagneticFluxReciprocal k_J=1
    "Josephson constant (k_J)";
  // The SI unit of magnetic flux (weber) is
  // inversely proportional to this value, which
  // should be increased for larger magnetic flux
  // numbers. Also, the SI unit of charge
  // (coulomb) is inversely proportional to this
  // value.
  constant Q.Resistance R_K=1
    "von Klitzing constant (R_K)";
  // The SI unit of electrical conductance
  // (siemen) is inversely proportional to this
  // value, which should be increased for larger
  // characteristic conductances. Also, the SI
  // unit of charge (coulomb) is inversely
  // proportional to this value.
  constant Q.RadiantIntensity 'cd'=1 "candela";
  // SI unit of luminous intensity
  constant Q.Number k_F=1 "Faraday constant (k_F)"
      ;
  // The unit of substance (mole) is inversely
  // proportional to this value, which should be
  // increased for larger particle numbers. If
  // k_F is set to 1, then charge is considered
  // to be an amount of substance.
  constant Q.Number R=1 "gas constant";
  // The unit of temperature (kelvin) is inversely
  // proportional to this value, which should be
  // increased for larger temperature numbers. If
  // R is set to 1, then temperature is
  // considered to be a potential.
end Basis;

record Am
  "Base constants and units for SI with k_F and R
      normalized instead of A and m"

  extends Basis(
    final R_inf=sqrt(8.3144621)*10973731.568539,
    final c=299792458/sqrt(8.3144621),
    final R_K=(96485.3365^2*25812.8074434)/8
        .3144621,
    final k_J=483597.870E9*sqrt(S*s)/m,
    final candela=1,
    final k_F=1,
    final R=1);
  // Note: The values of the un-normalized SI
  // base units are:
  //    A ~= 0.0000103643
  //    m ~= 0.346803
end Am;
```

4 Discussion and Conclusion

The implementation has been utilized to help model and simulate a proton exchange membrane fuel cell (PEMFC) in Dymola 7.4 [6]. It has been convenient in specifying the values of parameters and constants in this domain, where the product and research literature quotes values according to many different conventions. There are also cases where simulations have failed until the base constants were adjusted to properly scale critical values. In these cases, adjusting the `nominal` attributes of the variables did not seem to be sufficient, although it is difficult to prove.

The implementation raises the following concerns, which must be addressed in order to fully and consistently employ the method.

1. The `unit` attribute of a `Real` type should be renamed as `dimension` to indicate that it represents the physical dimension of the quantity rather than a particular unit.

2. In the new context, the `Real` type may be a misnomer. It may be best renamed as `Quantity`, but this may have implications on the name for the `Complex` record described in the Modelica language specification [12].

3. The `quantity` attribute of the `Real` type (possibly renamed as `Quantity`) may be superfluous. However, its removal may imply that the same attribute of the `Boolean`, `Integer`, and `String` types should be removed as well.

4. It would be helpful to establish a standard method to store and access the values of the base units and constants along with the results of a simulation. Ideally, the conversions created by the `defineUnitConversion` command (in Dymola) would be dynamically linked to the values of

the base units or constants, regardless of whether they are within an active model or from previous results.

5. The global variable `time` should be expressed as a quantity in the same manner as other variables—as the product of a number and a unit. Currently, `time` is time in unit seconds and the second has a value of 1. The `time` variable should be adjusted such that `time/U.s` is time in unit seconds and the second is not constrained to the value of 1. If the `der` operator is based on this unit-neutral time quantity, then it would be unnecesary to divide its output by the unit second (as in Sec. 3).

All of these items would affect both the Modelica language and the Modelica Standard Library. Therefore, it would be a rather significant undertaking to implement the method as a standard. However, not all of the items are necessary and the method can already be implemented to a limited extent (with work-arounds) in Modelica 3.2 and Dymola 7.4.

If a generalized method of units were to be introduced to Modelica, concepts from SysML may be pertinent and useful. Subsections C.4 and C.5 of version 1.2 of the SysML specification describe model libraries for “Quantity Kinds and Units” and “Quantities, Units, Dimensions, and Values” [1].

The proposed approach is not intended to supersede the previous work in unit checking in Modelica by Broman et al. ([4, 3]). Instead, it uses the methods of unit checking for dimension checking.

Acknowledgments

The authors wish to acknowledge support from the Presidential Fellowship of the Georgia Institute of Technology and the Robert G. Shackelford Fellowship of the Georgia Tech Research Institute.

References

[1] OMG Systems Modeling Language (OMG SysML®), Jun. 2010. Ver. 1.2.

[2] E. Allen, D. Chase, V. Luchangco, J.-W. Maessen, and G. L. S. Jr. Object-oriented units of measurement. In *OOPSLA04*, Vancouver, BC, Canada, Oct. 2004. ACM 1-58113-712-5/03/0010.

[3] P. Aronsson and D. Broman. Extendable physical unit checking with understandable error reporting. In *Proc. 7th Int. Modelica Conf.*, Como, Italy, Sep. 2009. Modelica Association.

[4] D. Broman, P. Aronsson, and P. Fritzson. Design considerations for dimensional inference and unit consistency checking in Modelica. In *Proc. 6th Int. Modelica Conf.*, Bielefeld, Germany, Mar. 2008. Modelica Association.

[5] Bureau International des Poids et Mesures. The International System of Units (SI). `http://www.bipm.org/en/si/si_brochure/`, Mar. 2006.

[6] K. L. Davies, C. J. Paredis, and C. L. Haynes. Library for first-principle models of proton exchange membrane fuel cells in Modelica. In *Proc. 9th Int. Modelica Conf.*, Munich, Germany, Sep. 2012 (accepted). Modelica Assoc.

[7] Dynasim AB. Dymola: Dynamic Modeling Laboratory, Mar. 2010. Ver. 7.4.

[8] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. IEEE Press, Piscataway, NJ, 2004.

[9] W. Greiner, L. Neise, and H. Stöcker. *Thermodynamics and statistical mechanics*. Classical theoretical physics. Springer-Verlag, 1995.

[10] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theory Comput.*, 4(3):435–447, 2008.

[11] S. E. Mattsson and H. Elmqvist. Unit checking and quantity conservation. In *Proc. 6th Int. Modelica Conf.*, University of Applied Sciences, Bielefeld, Germany, Mar. 2008. Modelica Assoc.

[12] Modelica Assoc. Modelica: A unified object-oriented language for physical systems modeling: Language specification. `https://www.modelica.org/documents/ModelicaSpec32.pdf`, Mar. 2010. Ver. 3.2.

[13] National Institute of Science and Technology. Fundamental physical constants—complete listing. `http://physics.nist.gov/cuu/Constants/Table/allascii.txt`, 2010. Accessed Jun. 2012.

[14] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 2nd edition, Apr. 2004.

[15] Wikipedia. Natural units. http://en.wikipedia.org/wiki/Natural_units. Accessed Mar. 2012.

DOI
10.3384/ecp12076801