Natural Unit Representation in Modelica

Kevin L. Davies and Christiaan J.J. Paredis
George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA USA

1. Introduction

The Modelica language establishes a formatted **`unit`** string for **`Real`** quantities. Methods for unit checking and unit inference are used [1, 2, 3]. Tools may support unit conversion for input and display, e.g., **`defineUnitConversion()`**.

Package Modelica is based on SI units [4]. However, other systems of units may be more convenient for certain applications. E.g., in electrochemistry, it is helpful to normalize the Faraday and gas constants [5].

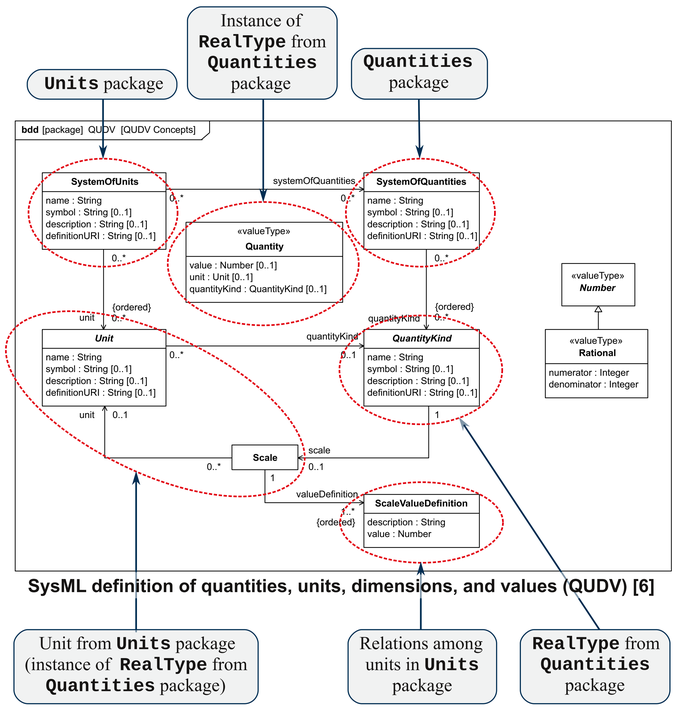
BIPM—the organization that maintains SI—states:

> "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." [4]

Although Modelica *tracks* units, it does not fully embrace this concept. The present work explores how this can be implemented in Modelica and the implications of such an approach.

2. Method

- Numeric systems of units—based on values assigned to a minimal set of base constants and interrelations among units



SysML definition of quantities, units, dimensions, and values (QUDV) [6]

Unit from **`Units`** package (instance of **`RealType`** from **`Quantities`** package)

Relations among units in **`Units`** package

`RealType` from **`Quantities`** package

3. Implementation

- Prototype of method coded in Modelica 3.2 and utilized with a fuel cell model library [5]

Listing 1: Excerpts from `Quantities` package (Q)

```
// Base quantities
type Angle = TypeReal(final unit="A");
type Length = TypeReal(final unit="L", min=0);
type Mass = TypeReal(final unit="M", min=0);
type Number = TypeReal(final unit="1");
type ParticleNumber = TypeReal(final unit="N", min=0) "Particle
  number";
type Time = TypeReal(final unit="T");

// Derived quantities
type MassSpecific = TypeReal(final unit="M/N") "Specific mass";
type Torque = TypeReal(final unit="L2.M/(A.T2)");
[...]
```

The **`unit`** attribute is used to denote the *dimension*.

Listing 3: Example `Base` record for a system of units

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

  final constant Q.Angle rad=1 "radian";
  constant Q.Wavenumber R_inf=sqrt(8.3144621)*10973731.568539
    "Rydberg constant";
  constant Q.Velocity c=299792458/sqrt(8.3144621) "speed of light
    in vacuum";
  constant Q.MagneticFluxReciprocal k_J=483597.870e9*sqrt(S*s)/m
    "Josephson constant";
  constant Q.ResistanceElectrical R_K=(96485.3365^2
    *25812.8074434)/8.3144621 "von Klitzing constant";
  constant Q.PowerRadiant 'cd'=1 "candela";
  constant Q.Number k_F=1 "Faraday constant";
  constant Q.Number R=1 "gas constant";
end Am;
```

Listing 2: Excerpts from `Units` package (U)

```
// Base physical constants and units
replaceable constant Bases.Am basis constrainedby Bases.Base
  "Scalable base constants and units";
final constant Q.Angle rad=basis.rad "radian";
[...]
final constant Q.Number R=basis.R "gas constant";

// Empirical units
constant Q.Length m=10973731.568539*rad/R_inf "meter";
constant Q.Time s=299792458*m/c "second";
constant Q.MagneticFlux Wb=483597.870e9/k_J "weber";
constant Q.ConductanceElectrical S=25812.8074434/R_K "siemen";
constant Q.ParticleNumber mol=96485.3365*Wb*S/k_F "mole";
constant Q.Potential K=8.3144621*(Wb*rad)^2*S/(s*mol*R) "kelvin";

// Remaining SI base units [BIPM2006, Table 1] and intermediates
final constant Q.Potential V=Wb*rad/s "volt";
final constant Q.Current A=V*S "ampere";
final constant Q.ParticleNumber C=A*s "coulomb";
final constant Q.Energy J=V*C "joule";
final constant Q.EnergyMassic Sv=(m/s)^2 "sievert";
final constant Q.Mass kg=J/Sv "kilogram ";

// SI prefixes [BIPM2006, Table 5]
final constant Q.Number yotta=1e24 "yotta (Y)";
[...]
final constant Q.Number yocto=1e-24 "yocto (y)";

// Coherent derived units in the SI [BIPM2006, Table 3]
final constant Q.Force N=J/m "newton";
final constant Q.Pressure Pa=N/m^2 "pascal";
[...]

// Non-SI units accepted for use with SI units [BIPM2006, Table 6]
final constant Q.Time min=60*s "minute";
[...]

// Derived physical constants
final constant Q.ConductanceElectrical G_0=2/R_K
  "conductance quantum";
final constant Q.Number alpha=pi*1e-7*c*s*G_0/(m*S)
  "fine-structure constant";
[...]
```

Base units may be chosen independently.

Most units are defined by interrelations.

SI prefixes are factors just like other units.

Other useful constants and non-SI units are included.

Listing 4: Example usage within a model

```
constant Q.MassSpecific m_H=1.00794*U.g/U.mol "Specific mass of
  hydrogen";
```

A quantity is the explicit product of a number and a unit.

Listing 5: Defining a unit conversion for display

```
defineUnitConversion("M/N", "g/mol", mol/g) "Specific mass";
```

This means: "Quantities with dimension mass per particle number may be displayed in g/mol after dividing by g/mol."

4. Results

Start-up time—once per session (Dymola 7.4, Ubuntu 11.10 (Linux), Intel Core 2 Duo):

- Translate units: 2.8 s
- Check unit relations (optional): 1.0 s
- Define unit conversions and default units: 2.7 s

Model translation and simulation time—1D transient thermal conduction and convection among 20 subregions (same platform):

- Translate: 18.8 s with "natural" units, 17.1 s without
- Simulate: 0.19 s with and without "natural" units

5. Discussion

- Start-up overhead is noticeable
  - Half of time is to re-translate units—unnecessary if base units have not changed
- Units are included in symbolic preprocessing
  - Overhead of ~10% during translation
  - No measureable effect on simulation time
- Existing framework for unit checking is appropriate for dimension checking [1,2 ,3]
  - Simpler because fewer fundamental dimensions than SI units
- Work-arounds necessary in Modelica 3.2:
  - **`der()`** operator must be divided by **`U.s`**
  - **`time`** variable must be multiplied by **`U.s`**

6. Conclusion

Summary: Modelica can express physical values in a manner that is unit-neutral by fully embracing the concept of a physical value as the product of a number and a unit [4].

Advantages:

- Consistent with the essence of quantities, values, units, and numbers [4]
- Supports non-SI unit systems (CGS, Planck, imperial, etc.)
- Units from multiple unit systems can be used in the same model (where compatible)
- Selected physical constants can be normalized

Disadvantages and limitations:

- Unfamiliar way of thinking
- Not used in Modelica Standard Library
- Overhead during start-up and model translation
- Only affine units are directly supported
- Other tools must correctly interpret simulation results (e.g., a value of 1 for velocity may not be 1 m/s)

Acknowledgements

- Robert G. Shackelford Fellowship from Georgia Tech Research Institute
- Presidential Fellowship from George W. Woodruff School of Mechanical Engineering

References

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

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

[3] S. Mattsson and H. Elmqvist. Unit checking and quantity conservation. In *Proc. 6th Int. Modelica Conf.*, 2008.

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

[5] K. Davies, C. Paredis and C. Haynes. Library for first-principle models of proton exchange membrane fuel cells in Modelica. In *Proc. 9th Int. Modelica Conf.*, 2012.

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