Note (2009-07-27): This is a proposed addition to the SysML specification (http://www.omg.org/spec/SysML/1.1). It is currently being considered by the SysML Revision Task Force (RTF). If accepted, it will become a non-normative annex of the SysML 1.2 specification.

C.5 Model Library for Quantities, Units, Dimensions and Values (QUDV)

C.5.1 Overview

For any system model, a solid foundation of well-defined quantities, units and dimensions system is very important. Properties that describe many aspects of a system depend on it. At the same time, such a foundation should be a shareable resource that can reused in many models within and across organizations and projects.

The most widely accepted, scrutinized and globally used system of quantities and system of units are the International System of Quantities (ISQ) and the International System of Units (SI). They are formally standardized through [ISO31] and [IEC60027]. Currently, work is in progress to harmonize these two sets of standards into one new set [ISO/IEC80000], of which a number of parts are already released with the remaining parts expected to be completed in the near future. All the relevant concepts underlying ISQ and SI are publicly available in [VIM]. See Section C.5.3 for references to these documents.

At a minimum, SysML should provide the means to support the imminent international standard [ISO/IEC80000]. In addition, many other systems of quantities and units are still in use for particular applications and for historical reasons. A prime example is the system based on UK Imperial units, which are still widely used in North America. SysML should provide the means to support all such specific systems of quantities and units, including precise definitions of the relationships between different systems of units, and with explicit and unambiguous unit conversions to and from SI as well as other systems.

To provide a solid and stable foundation, the model for defining quantities, units, dimensions and values in SysML is explicitly based on the concepts defined in [VIM], which have been written by the authoritative Working Group 2 of the Joint Committee for Guides in Metrology (JCGM/WG 2), in which the JCGM member organizations are represented: BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP and OIML. At the same time, the model library is designed in such a way that extensions to the ISQ and SI can be represented, as well as any alternative systems of quantities and units.

The model library can be used to support SysML user models in various ways. A simple approach is to define and document libraries of reusable systems of units and quantities for reuse across multiple projects, and to link units and quantity kinds from these libraries to Unit and QuantityKind stereotypes defined in SysML user models. The name of a Unit or QuantityKind stereotype, its definitionURI, or other means may be used to link it with definitions made using this library. Instances of blocks conforming to this model library may be created by instance specifications, as shown in Section C.5 below, or by other means.

Even though this model library is specified in terms of SysML blocks, its contents could equally be specified in terms of UML classes without dependencies on any SysML extensions. This Annex specifies the model library using SysML blocks to maintain compatibility with the SysML specification. UML and other versions of this same conceptual model are also important and useful to align different standards with each other and with those of VIM.

Separate serializations of this model library, including a UML model generated as a simple transformation from the model library specified in this Annex, together with other supporting resources such as examples and reference libraries of systems of units and quantities built using this model, are expected to be published via the SysML Project Portal wiki at http://www.omgwiki.org/OMGSysML/.

C.5.2 Abstract Syntax

Figures C.8-C.10 present the QUDV model library in a series of diagrams.

The QUDV Concepts diagram in Figure C.8 presents the core concepts of System of Units, Unit, SystemOfQuantities, and QuantityKind. Distinguished unit values can be organized in a measurement scale for a particular QuantityKind.A Quantity of a given QuantityKind contains a numerical value expressed in a particular measurement unit. Subtypes of Number include Integer, Real, and Complex as defined in the SysML Blocks model library specified in Chapter 8, Blocks. A Quantity type may optionally be used to type properties or other values.

In the QUDV Unit diagram in Figure C.9, SimpleUnit provides the basis for defining other units via conversion or derivation. Additionally, QUDV provides support for specifying a coherent derived unit as a product of the baseUnit(s) of a given SystemOfUnits. In a coherent SystemOfUnits, there is only one base unit for each base quantity kind.

In the QUDV QuantityKind diagram in Figure C.10, SimpleQuantityKind provides the basis for defining other quantity kinds via specialization or derivation. QUDV provides a declarative specification of dimensional analysis to assign to each QuantityKind an expression of its dependence on the baseQuantityKind(s) of a SystemOfQuantities. This dependence is expressed as a product of powers of DimensionFactor(s) corresponding to the base quantities. Section C.5.3.20, “SystemOfQuantities” specifies the derivation of quantity dimensions using an algorithm specified in OCL.

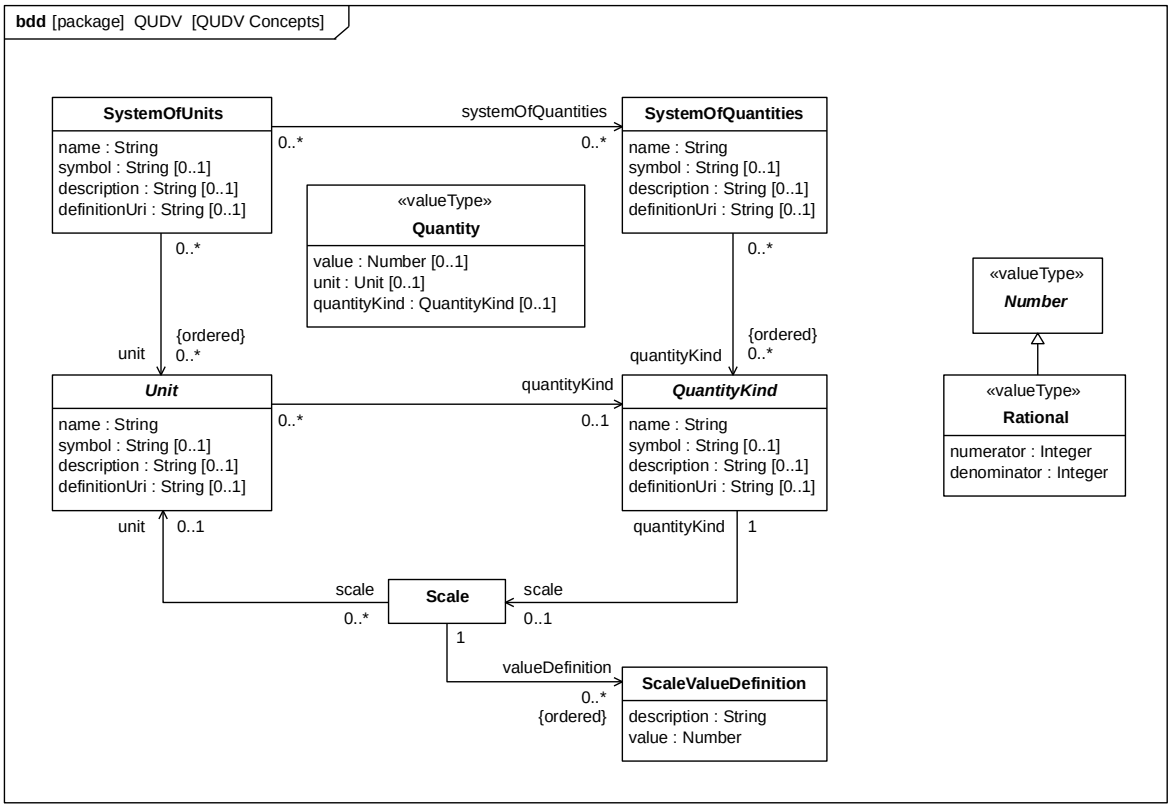

Figure C.8 – QUDV Concepts diagram

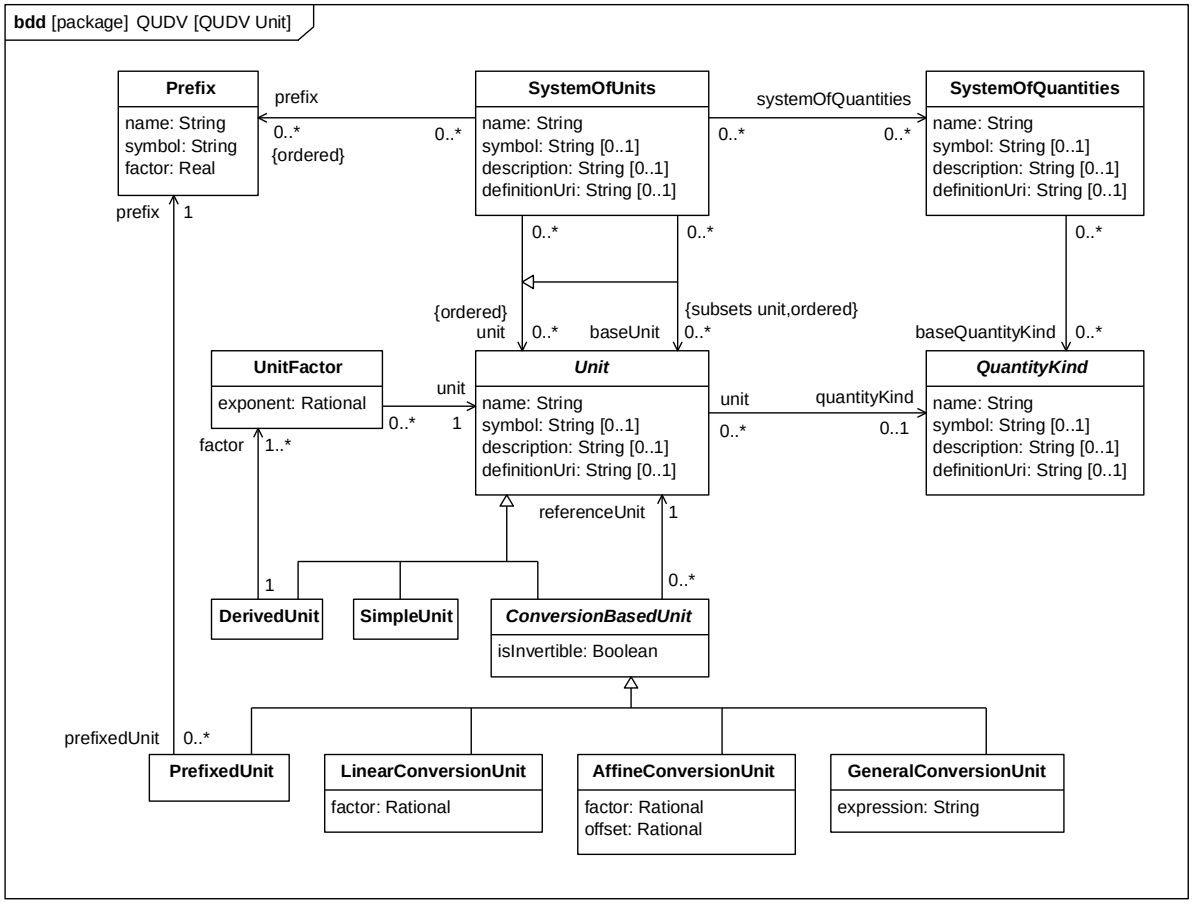

Figure C.9 – QUDV Unit diagram

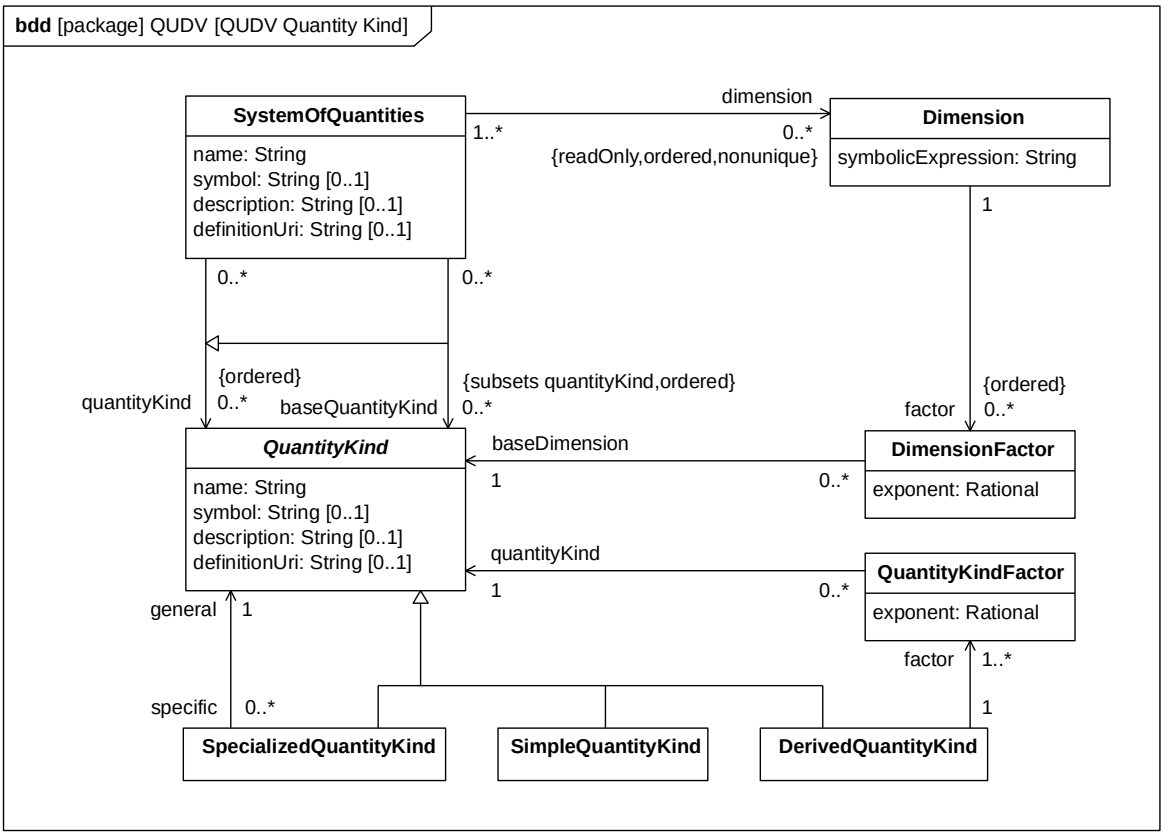

Figure C.10 – QUDV QuantityKind diagram

C.5.3.1 AffineConversionUnit

Description

An AffineConversionUnit is a ConversionBasedUnit that represents a measurement unit that is defined with respect to another reference measurement unit through an affine conversion relationship with a conversion factor and offset.

The unit conversion relationship is defined by the following equation:

$$value_{RU} = factor \cdot value_{CU} + offset$$

where:

$value_{RU}$ is the quantity value expressed in the referenceUnit, and,
$value_{CU}$ is the quantity value expressed in the AffineConversionUnit.

E.g. in the definition of the AffineConversionUnit for “degree Fahrenheit” with respect to the referenceUnit “degree Celsius”, the factor would be 5/9 and the offset would be -160/9, because $T_{Celsius} = 5/9 \cdot T_{Fahrenheit} - 160/9$ which is equivalent with $T_{Fahrenheit} = 9/5 \cdot T_{Celsius} + 32/1$.

Properties

- factor: Rational
  Rational number that specifies the factor in the unit conversion relationship.

- offset: Rational
  Rational number that specifies the offset in the unit conversion relationship.

C.5.3.2 ConversionBasedUnit

Description

A ConversionBasedUnit is an abstract classifier that is a Unit that represents a measurement unit that is defined with respect to another reference unit through an explicit conversion relationship.

Properties

- referenceUnit: Unit
  Specifies the unit with respect to which the ConversionBasedUnit is defined.
- isInvertible: Boolean
  Specifies whether the unit conversion relationship is invertible. For LinearConversionUnit and AffineConversionUnit this is always true.

C.5.3.3 DerivedQuantityKind

Description

A DerivedQuantityKind is a QuantityKind that represents a kind of quantity that is defined as a product of powers of one or more other kinds of quantity. A DerivedQuantityKind may also be used to define a synonym kind of quantity for another kind of quantity.

For example “velocity” can be specified as the product of “length” to the power one times “time” to the power minus one, and subsequently “speed” can be specified as “velocity” to the power one.

Properties

- factor: QuantityKindFactor [1..*]
  Set of QuantityKindFactor that specifies the product of powers of other kind(s) of quantity that define the DerivedQuantityKind.

C.5.3.4 DerivedUnit

Description

A DerivedUnit is a Unit that represents a measurement unit that is defined as a product of powers of one or more other measurement units.

For example the measurement unit “metre per second” for “velocity” is specified as the product of “metre” to the power one times “second” to the power minus one.

Properties

- factor: UnitFactor [1..*]
  Set of UnitFactor that specifies the product of powers of other measurement units that define the DerivedUnit.

C.5.3.5 Dimension

Description

A Dimension represents the [VIM] concept of “quantity dimension” that is defined as “expression of the dependence of a quantity on the base quantities of a system of quantities as a product of powers of factors corresponding to the base quantities, omitting any numerical factor”.

For example in the ISQ the quantity dimension of “force” is denoted by dim F = L·M·T2, where “F” is the symbol for “force”, and “L”, “M”, “T’ are the symbols for the ISQ base quantities “length”, “mass” and “time” respectively.

The Dimension of any QuantityKind can be derived through the algorithm that is defined in C.5.3.20 with SystemOfQuantities. The actual Dimension for a given QuantityKind depends on the choice of baseQuantityKind specified in a SystemOfQuantities.

Properties

- symbolicExpression: String [0..1]
  Symbolic expression of the quantity dimension’s product of powers, in terms of symbols of the kinds of quantity that represent the base kinds of quantity and their exponents.
- factor: DimensionFactor [0..*] {ordered}
  Ordered set of DimensionFactor that specifies the product of powers of base dimensions that define the Dimension. The possible base dimensions are represented by the ordered set of baseQuantityKind defined in the SystemOfQuantities for which the Dimension is specified. The order of the factors should follow the ordered set of baseQuantityKind in SystemOfQuantities.

C.5.3.6 DimensionFactor

Description

A DimensionFactor represents a factor in the product of powers that defines a Dimension.

Properties

- exponent: Rational
  Rational number that specifies the exponent of the power to which the baseDimension is raised.
- baseDimension: QuantityKind
  Reference to the QuantityKind that represents the base quantity dimension in the factor.

C.5.3.7 GeneralConversionUnit

Description

A GeneralConversionUnit is a ConversionBasedUnit that represents a measurement unit that is defined with respect to another reference measurement unit through a conversion relationship expressed in some syntax through a general mathematical expression.

The unit conversion relationship is defined by the following equation:
valueRU / valueCU = f(valueRU, valueCU), where:
valueRU is the quantity value expressed in the referenceUnit, and,
valueCU is the quantity value expressed in the GeneralConversionUnit, and,
f(valueRU, valueCU) is a mathematical expression that includes valueRU and valueCU.

Properties

- expression: String
  Specifies the unit conversion relationship in some expression syntax.
- expressionLanguageUri: UriString [0..1]
  URI that specifies the language for the expression syntax.

C.5.3.8 LinearConversionUnit

Description

A LinearConversionUnit is a ConversionBasedUnit that represents a measurement unit that is defined with respect to another measurement reference unit through a linear conversion relationship with a conversion factor.

The unit conversion relationship is defined by the following equation:
valueRU = factor · valueCU, where:
valueRU is the quantity value expressed in the referenceUnit, and,
valueCU is the quantity value expressed in the LinearConversionUnit.
E.g. in the definition of the LinearConversionUnit for "inch" with respect to the referenceUnit "metre", the factor would be 254/10000, because 0.0254 metre = 1 inch.

Properties

- factor: Rational
  Rational number that specifies the factor in the unit conversion relationship.

C.5.3.9 Prefix

Description

A Prefix represents a named multiple or submultiple multiplication factor used in the specification of a PrefixedUnit. A SystemOfUnits may specify a set of prefixes.

Properties

- name: String
  Name of the prefix.
- symbol: String [0..1]
  Short symbolic name of the prefix.
- factor: Real [1]
  Specifies the multiple or submultiple multiplication factor.

C.5.3.10 PrefixedUnit

Description

A PrefixedUnit is a ConversionBasedUnit that represents a measurement unit that is defined with respect to another measurement reference unit through a linear conversion relationship with a named prefix that represents a multiple or submultiple of a unit.

[VIM] defines "multiple of a unit" as "measurement obtained by multiplying a given measurement unit by an integer greater than one" and "submultiple of a unit" as "measurement unit obtained by dividing a given measurement unit by an integer greater than one".

The unit conversion relationship is defined by the following equation:
valueRU = factor · valueCU, where:
valueRU is the quantity value expressed in the referenceUnit, and,
valueCU is the quantity value expressed in the PrefixedUnit.
E.g. in the definition of the PrefixedUnit for "megabyte" with respect to the referenceUnit "byte", the prefix would be the Prefix for "mega" with a factor 106, because 106 byte = 1 megabyte.

See [VIM] for all decimal and binary multiples and decimal submultiples defined in SI.

Properties

- prefix: Prefix
  Specifies the prefix that defines the name, symbol and factor of the multiple or submultiple.

Constraints

[1] The referenceUnit shall not be a PrefixedUnit, i.e. it is not allowed to prefix an already prefixed measurement unit. In general the referenceUnit should be a SimpleUnit.

```
package QUDV
context PrefixedUnit
```

```
inv: not referenceUnit->oclIsTypeOf(PrefixedUnit)
endpackage
```

C.5.3.11 Quantity

Description

A Quantity is a UML DataType or SysML ValueType that represents the [VIM] concept of “quantity” that is defined as “property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference”. In SysML it specifies the type of a value property.

Properties

- quantityKind: QuantityKind [0..1]
  Specification of the associated QuantityKind.
- unit: Unit [0..1]
  Specification of the associated Unit.
- value: Number [0..1]
  Specification of the associated Value.

Constraints

[1] At least one of the three properties shall be specified.

```
package QUDV
context Quantity
inv: quantityKind->isNotEmpty()
  or unit->isNotEmpty()
  or value->isNotEmpty()
endpackage
```

C.5.3.12 QuantityKind

Description

A QuantityKind is an abstract classifier that represents the [VIM] concept of “kind of quantity” that is defined as “aspect common to mutually comparable quantities”. A QuantityKind represents the essence of a quantity without any numerical value or unit. Quantities of the same kind within a given system of quantities have the same quantity dimension. However, quantities of the same dimension are not necessarily of the same kind.

Properties

- name: String
  Name of the kind of quantity.
- symbol: String [0..1]
  Short symbolic name of the kind of quantity.
- description: String [0..1]
  Textual description of the kind of quantity.
- definitionUri: UriString [0..1]
  URI that references an external definition of the kind of quantity.
- scale: Scale [0..1]
  Specification of a Scale that is associated to the QuantityKind.

C.5.3.13 QuantityKindFactor

Description

A QuantityKindFactor represents a factor in the product of powers that defines a DerivedQuantityKind.

Properties

- exponent: Rational
  Rational number that specifies the exponent of the power to which the quantityKind is raised.
- quantityKind: QuantityKind
  Reference to the QuantityKind that participates in the factor.

C.5.3.14 8.3.3.2 Rational

Description

A Rational value type represents the mathematical concept of a number that can be expressed as a quotient of two integers. It may be used to express the exact value of such values, without issues of rounding or other approximations if the result of the division were used instead.

Properties

- numerator: Integer
  An integer number used to express the numerator of a rational number.
- denominator: Integer
  An integer number used to express the denominator of a rational number.

C.5.3.15 Scale

Description

A Scale represents the [VIM] concept of a “measurement scale” that is defined as an “ordered set of quantity values of quantities of a given kind of quantity used in ranking, according to magnitude, quantities of that kind”. A Scale specifies one or more fixed values that have a specific significance in the definition of the associating QuantityKind.

For example the “thermodynamic temperature” kind of quantity is defined by specifying the values of 0 and 273.16 kelvin as the temperatures of absolute zero and the triple point of water respectively.

A Scale does not always need to specify a unit. For example the “Rockwell C Hardness Scale” or the “Beaufort Wind Force Scale” are ordinal scales that do not have a particular associated unit. Similarly, subjective scales for a “priority” or “risk” kind of quantity with e.g. value definitions 0 for “low”, 1 for “medium” and 3 for “ high” do not have a particular associated unit.

Properties

- valueDefinition: ScaleValueDefinition [1..*] {ordered}
  Ordered set of ScaleValueDefinition that specifies the defined numerical value(s) and textual definition(s) for the measurement scale.
- unit: Unit [0..1]
  Optionally specifies the unit in which the value of each valueDefinition is expressed.

Constraints

[1] If a unit is specified, then it shall be the same as the unit of the associating QuantityKind.

```
package QUDV
context Scale
inv: unit->isEmpty() or unit.quantityKind = self.quantityKind
endpackage
```

C.5.3.16 ScaleValueDefinition

Description

A ScaleValueDefinition represents a specific value for a measurement scale.

Properties

- value: Number
  Specifies the numerical value.
- definition: String
  Specifies the textual definition for the value.

C.5.3.17 SimpleQuantityKind

Description

A SimpleQuantityKind is a QuantityKind that represents a kind of quantity that does not depend on any other QuantityKind. Typically a base quantity would be specified as a SimpleQuantityKind.

C.5.3.18 SimpleUnit

Description

A SimpleUnit is a Unit that represents a measurement unit that does not depend on any other Unit. Typically a base unit would be specified as a SimpleUnit.

C.5.3.19 SpecializedQuantityKind

Description

A SpecializedQuantityKind is a QuantityKind that represents a kind of quantity that is a specialization of another kind of quantity.

For example, “distance”, “width”, “depth”, “radius” and “wavelength” can all be specified as specializations of the “length” SimpleQuantityKind.

Properties

- general: QuantityKind
  Specifies the QuantityKind that is specialized.

C.5.3.20 SystemOfQuantities

Description

A SystemOfQuantities represents the [VIM] concept of “system of quantities” that is defined as a “set of quantities together with a set of non-contradictory equations relating those quantities”. It collects a list of QuantityKind that specifies the kinds of quantity that are known in the system.

The International System of Quantities (ISQ) is an example of a SystemOfQuantities, defined in [ISO31] and [ISO/IEC80000].

Properties

- name: String
  Name of the system of quantities.
- symbol: String [0..1]
  Short symbolic name of the system of quantities.
- description: String [0..1]
  Textual description of the system of quantities.
- definitionUri: UriString [0..1]
  URI that references an external definition of the system of quantities.
  Note: As part of [ISO/IEC80000] normative URIs for each of the ISQ quantities and SI units are being defined.

- quantityKind: QuantityKind [0..*] {ordered}
  Ordered set of QuantityKind that specifies the kinds of quantity that are known in the system.
- baseQuantityKind: QuantityKind [0..*] {ordered, subsets quantityKind}
  Ordered set of QuantityKind that specifies the base quantities of the system of quantities. This is a subset of the complete quantityKind list. The base quantities define the basis for the quantity dimension of a kind of quantity.
- /dimension: Dimension [0..*] {ordered, readOnly, nonunique}
  Derived ordered set of Dimension. The actual dimension of a QuantityKind depends on the list of baseQuantityKind that are specified in an actual SystemOfQuantities, see the DerivedDimensions constraint.

Constraints

[1] All quantity dimensions are derived through the following algorithm specified in OCL.

```
package QUDV

-- Some constraints below use a non-standard closure operator available
-- in the Eclipse MDT OCL implementation of the OMG OCL 2.0 specification.
-- http://www.eclipse.org/modeling/mdt/?project=ocl

-- get the set of units, if any, that a given unit directly depends on
context Unit
def: directUnitDependencies : Set(Unit)
 = if oclIsKindOf(ConversionBasedUnit)
 then oclAsType(ConversionBasedUnit).referenceUnit->asSet()
 else
   if oclIsKindOf(DerivedUnit)
   then oclAsType(DerivedUnit).factor.unit->asSet()
   else Set{}
   endif
 endif

-- get the set of units, if any, that a given unit direclty or indirectly depends on
context Unit
def: allUnitDependencies : Set(Unit)
 = self->asSet()->closure(directUnitDependencies)

context Unit
inv acyclic_unit_dependencies
 : allUnitDependencies->excludes(self)

-- get the set of quantityKinds, if any, that a given quantityKind directly depends on
context QuantityKind
def: directQKindDependencies : Set(QuantityKind)
 = if oclIsKindOf(DerivedQuantityKind)
 then oclAsType(DerivedQuantityKind).factor.quantityKind->asSet()
 else
   if oclIsKindOf(SpecializedQuantityKind)
   then oclAsType(SpecializedQuantityKind).general->asSet()
   else Set{}
   endif
 endif

context QuantityKind
def: allQuantityKindDependencies : Set(QuantityKind)
 = self->asSet()->closure(directQKindDependencies)

context QuantityKind
inv acyclic_quantity_kind_dependencies
 : allQuantityKindDependencies->excludes(self)

context SystemOfQuantities::deriveQuantityKindDimensions() :
post: quantityKind->forAll(qK|qK.hasProperDimension(self))

-- The derived dimension of a simple quantity kind must have exactly one factor
-- whose numerator and denominator are equal to 1.
```

```
context SimpleQuantityKind
def: hasProperDimension(sq:SystemOfQuantities) : Boolean
 = let d:Dimension=sq.getDimension(self) in d.factor->size()=1 and d.factor.exponent-
>forAll(numerator=1 and denominator=1)

-- The derived dimension of a specialized quantity kind is
-- the dimension of its general quantity kind.
context SpecializedQuantityKind
def: hasProperDimension(sq:SystemOfQuantities) : Boolean
 = sq.getDimension(self) = sq.getDimension(general)

-- A helper function to produce the factor/quantityKind tuples for a given Dimension.
context Dimension
def: dimFactors : Bag(Tuple(factor:qudv::blocks::Rational,qKind:QuantityKind))
= self.factor->collect(df:DimensionFactor|
       Tuple{factor=df.exponent,qKind=df.baseDimension})

-- A helper function to get all the factor/quantityKind tuples
-- for the dimension factors of a derived quantity kind.
context DerivedQuantityKind
def: derQFactors(sq:SystemOfQuantities) :
Bag(Tuple(factor:qudv::blocks::Rational,qKind:QuantityKind))
 = self.factor->collect(qf:QuantityKindFactor| let qd:Dimension =
sq.getDimension(qf.quantityKind) in
    qd.factor->collect(df:DimensionFactor|
Tuple{factor=qf.exponent.plus(df.exponent),qKind=qf.quantityKind}))

-- Reduce a bag of factor/quantityKind tuples by combining all factors for the same
quantity kind
-- and eliminating the zero-factor/quantityKind tuples
context DerivedQuantityKind
def: reducetoNonZeroUniqueFactors(
   qFactors:Bag(Tuple(factor:qudv::blocks::Rational,qKind:QuantityKind)),
   qKinds:Set(QuantityKind))
: Bag(Tuple(factor:qudv::blocks::Rational,qKind:QuantityKind))
 = let uqFactors:Bag(Tuple(factor:qudv::blocks::Rational,qKind:QuantityKind))
   = qKinds->collect(
     -- for each unique quantity kind, qKind1, from the set of unique quantity kinds,
qKinds...
     qKind1:QuantityKind|
     -- get the sequence of factors from the set of qFactors tuples whose quantity kind
is qKind1...
     let factor1s:Sequence(qudv::blocks::Rational)
     = qFactors->select(qKind=qKind1)->collect(factor)->asSequence()
     -- start with the first factor, factor1, from all the factor1s associated to
qKind1...
     in let factor1:qudv::blocks::Rational=factor1s->first()
       -- construct the factor/quantityKind tuple for qKind1 where
       -- the factor is the product of factor1 with all remaining factors1s
       in Tuple{
         factor=factor1s->excluding(factor1)->iterate(
           factorI:qudv::blocks::Rational;
           factorN:qudv::blocks::Rational=factorI | factorN.plus(factorI)),
         qKind=qKind1})
-- eliminate the factor/quantityKind tuples where the factor is zero
in let nqFactors:Bag(Tuple(factor:qudv::blocks::Rational,qKind:QuantityKind))
   = uqFactors->select(factor.numerator<>0)
in nqFactors

-- The derived dimension of a derived quantity kind is the simplified set of
factor/quantityKind tuples
-- for the derived quantity kind. The simplified set of factor/quantityKind tuples has
-- one factor/quantityKind tuple for each quantityKind where the simplified factor
-- is a non-zero product of all the factors in the factor/quantityKind tuples.
context DerivedQuantityKind
def: hasProperDimension(sq:SystemOfQuantities) : Boolean
 = let d:Dimension = sq.getDimension(self)
in let resFactors : Bag(Tuple(factor:qudv::blocks::Rational,qKind:QuantityKind))
 = d.dimFactors

-- the unique quantityKinds from the result...
```

```
in let resKinds:Set(QuantityKind)=resFactors->collect(qKind)->asSet()

-- the factor/quantityKind tuples from the derived quantity...
in let qFactors:Bag(Tuple(factor:qudv::blocks::Rational,qKind:QuantityKind))
 = self.derQFactors(sq)

-- the unique quantityKinds from the derived quantity...
in let qKinds:Set(QuantityKind)=qFactors->collect(qKind)->asSet()

-- get the reduced non-zero factor/quantityKinds...
in let nqFactors:Bag(Tuple(factor:qudv::blocks::Rational,qKind:QuantityKind))
 = self.reducetoNonZeroUniqueFactors(qFactors, qKinds)

-- condition1: there should be the same number of factor/quantityKind tuples in the
result
-- compared to the non-zero unique factor/quantityKind tuples for the derivedQuantityKind
in nqFactors->size() = resFactors->size()

-- condition2: there should be the same set of quantity kinds in the result
-- and in the non-zero unique factor/quantityKind tuples
and qKinds->symmetricDifference(resKinds)->isEmpty()

-- condition3: for each quantity kind, the factors in the result and
-- in the reduced non-zero unique factor/quantityKind tuples should be equivalent
rationals
and qKinds->forAll(qk:QuantityKind|
   let nFactor:qudv::blocks::Rational=nqFactors->select(qKind=qk)->collect(factor)-
>asSequence()->first()
in let rFactor:qudv::blocks::Rational=resFactors->select(qKind=qk)->collect(factor)-
>asSequence()->first()
in nFactor.equivalent(rFactor))

endpackage
```

C.5.3.21 SystemOfUnits

Description

A SystemOfUnits represents the [VIM] concept of “system of units” that is defined as “set of base units and derived units, together with their multiples and submultiples, defined in accordance with given rules, for a given system of quantities”. It collects a list of Unit that are known in the system. A SysML SystemOfUnits only optionally defines multiples and submultiples.

Properties

- name: String
  Name of the system of units.
- symbol: String [0..1]
  Short symbolic name of the system of units.
- description: String [0..1]
  Textual description of the system of units.
- definitionUri: UriString [0..1]
  URI that references an external definition of the system of units.
  Note: As part of [ISO/IEC80000] normative URIs for each of the quantities in the ISQ and units in the SI are being defined.
- unit: Unit [0..*] {ordered}
  Ordered set of Unit that specifies the units that are known in the system.
- baseUnit: Unit [0..*] {ordered, subsets unit}
  Ordered set of Unit that specifies the base units of the system of units. A “base unit” is defined in [VIM] as a “measurement unit that is adopted by convention for a base quantity”, i.e. it is the (preferred) unit in which base quantities of the associated systemOfQuantities are expressed.

- prefix: Prefix [0..*] {ordered}
  Ordered set of Prefix that specifies the prefixes for multiples and submultiples of units in the system.
- systemOfQuantities: SystemOfQuantities [0..1]
  Reference to the SystemOfQuantities for which the units are specified.

Constraints

[1] In a coherent system of units, there is only one base unit for each base quantity.

```
package QUDV
context SystemOfUnits
def: isCoherent() : qudv::blocks::Boolean =
  baseUnit->size() = systemOfQuantities.baseQuantityKind->size()
  and baseUnit->forAll(bU|
       systemOfQuantities.baseQuantityKind->one(bQK|bU.quantityKind=bQK))
  and systemOfQuantities.baseQuantityKind->forAll(bQK|
       baseUnit->one(bU|bQK=bU.quantityKind))
endpackage
```

[2] A coherent derived unit is a derived unit that, for a given system of quantities and for a chosen set of base units, is a product of powers of base units with no other proportionality factor than one.

```
package QUDV
context SystemOfUnits
def: isCoherent(du : DerivedUnit) : qudv::blocks::Boolean =
  baseUnit->includesAll(du.factor.unit)
  and du.factor.exponent->forAll(numerator=1 and denominator=1)
endpackage
```

C.5.3.22 Unit

Description

A Unit is an abstract classifier that represents the [VIM] concept of “measurement unit” that is defined as “real scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of the two quantities as a number”.

Properties

[1] name: String
Name of the unit.

[2] symbol: String [0..1]
Short symbolic name of the unit.

[3] description: String [0..1]
Textual description of the unit.

[4] definitionUri: UriString [0..1]
URI that references an external definition of the unit.

C.5.3.23 UnitFactor

Description

A UnitFactor represents a factor in the product of powers that defines a DerivedUnit.

Properties

[1] exponent: Rational
Rational number that specifies the exponent of the power to which the unit is raised.

[2] unit: Unit
Reference to the Unit that participates in the factor.

C.5.3 References

[1] [VIM]
JCGM 200:2008, International Vocabulary of Metrology – Basic and General Concepts and Associated Terms (VIM), 3rd edition, 2008, BIPM, Paris, France.
Available for download in PDF format from BIPM's website http://www.bipm.org.
This third edition is also published on paper by ISO (ISO/IEC Guide 99-12:2007, International Vocabulary of Metrology — Basic and General Concepts and Associated Terms, VIM)

[2] [ISO/IEC80000]
ISO/IEC 80000, Quantities and units, (15 parts, some published, some still in progress, harmonized replacement of [ISO31] and [IEC60027], the new international system of quantities and units)

[3] [ISO31]
ISO 31, Quantities and units (Third edition 1992-08-01) (Specifies the international system of units – SI – in 14 parts)

[4] [IEC60027]
IEC 60027-2:2005, Letter symbols to be used in electrical technology – Part 2: Telecommunications and electronics (Third edition 2005-08)

[5] [SI-Brochure]
Le Système international d'unités (SI) / The International System of Units (SI), 8th edition 2006, BIPM, (French and English)
Available for download in PDF format from http://www.bipm.org/en/si/si_brochure .

[6] [NIST330]
The International System of Units (SI),
NIST Special Publication 330, 2008 Edition
NOTE: U.S. version of the English language text of [SI-Brochure]
Available for download in PDF format from http://physics.nist.gov/cuu/Units/bibliography.html

[7] [NIST822]
Guide for the Use of the International System of Units (SI), NIST Special Publication 811, 2008 Edition
Available for download in PDF format from http://physics.nist.gov/cuu/Units/bibliography.html

C.5.4 Usage Examples

C.5.4.1 SI Unit and QuantityKind examples

Figure C.11 shows an approach for defining the seven base units of the System International of Units defined in http://www.bipm.org/en/si/si_brochure/chapter2/2-1/ and http://physics.nist.gov/cuu/Units/units.html. This approach involves instantiating the concrete classes of Unit shown in Figure C.9. For example, “second” is defined as a SimpleUnit (C.5.3.18) which, in the context of the “SI” SystemOfUnits (C.5.3.21) is both a unit and a base unit. The same instantiation technique applies to define the seven base quantity kinds of the International System of Quantities (ISQ) corresponding to the seven base units of the SI.

Figure C.12 diagram shows the definition of “newton” as a DerivedUnit (C.5.3.4) corresponding to the “forceQK” DerivedQuantityKind (C.5.3.3). Derived units and quantity kinds are defined as products of factors on other units and quantity kinds respectively. In the QUDV, the product factors of a DerivedUnit (resp. DerivedQuantityKind) are all of the UnitFactor (resp. DerivedUnitFactor) at the “factor” ends of association link instances.

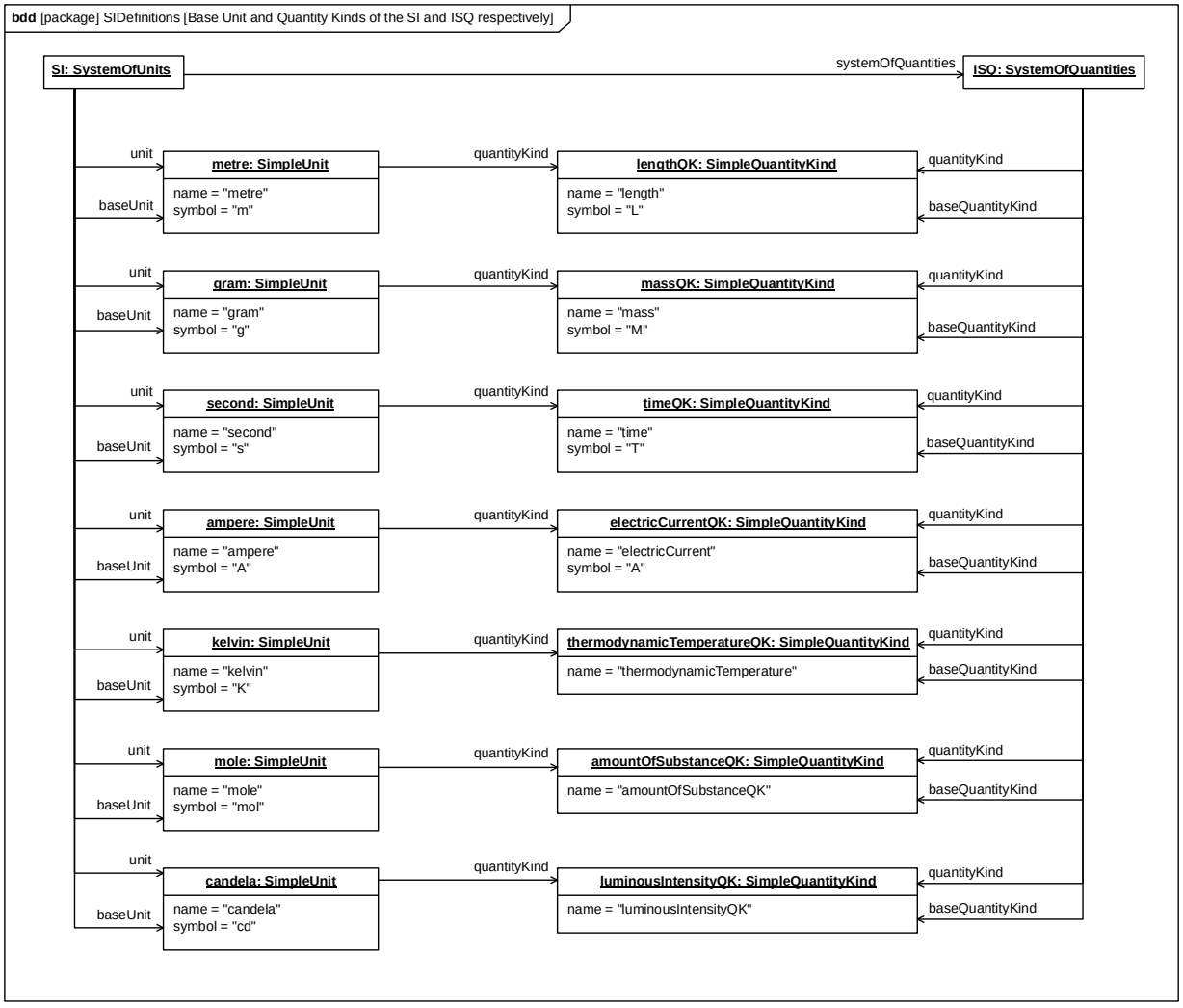

Figure C.11 – Base Unit and Quantity Kinds of the SI and ISQ respectively

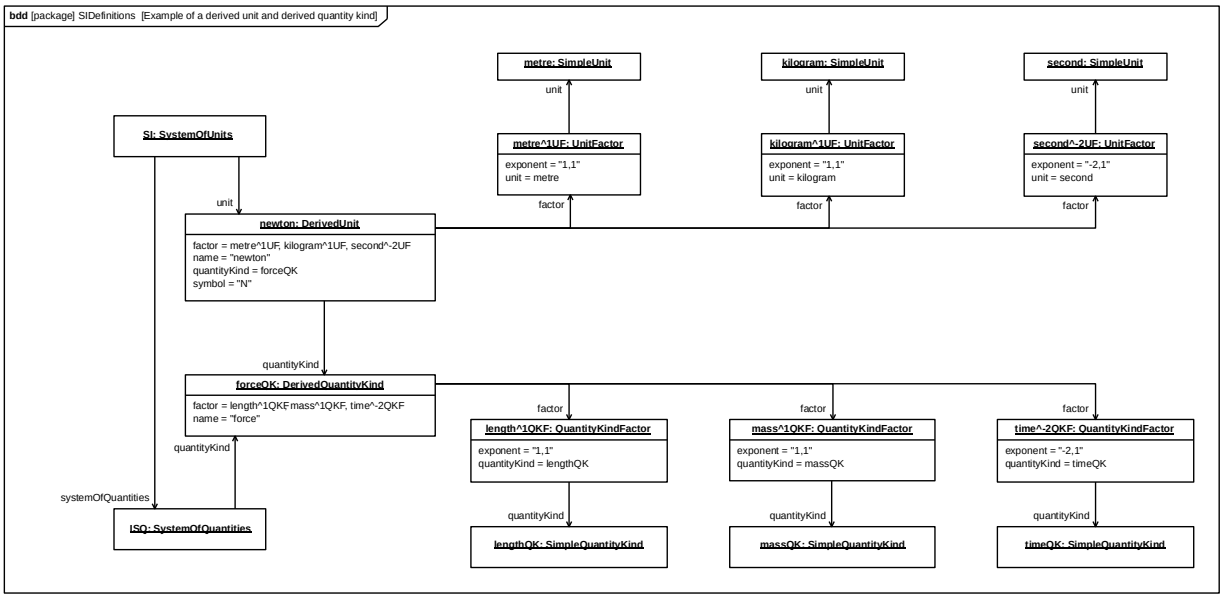

Figure C.12 – Example of a derived unit and derived quantity kind

C.5.4.2 Spring example

Figure C.13 shows a simple model of the length of a spring defined as the linear distance between the linear position of its two flange ends. QUDV supports defining arbitraty systems of units and quantities. Although this example uses only one unit, “metre” and one quantity kind, “lengthQK”; this example illustrates support for specializing the concept of Quantity to make additional distinctions such as “LinearPosition” vs. “LinearDistance”, two distinct quantities that have the same unit and quantity kind. This example illustrates an instance of a spring and uses the dot pathname property notation defined for IBD’s (8.3.1.2) to clearly indicate the role of each instance specification.

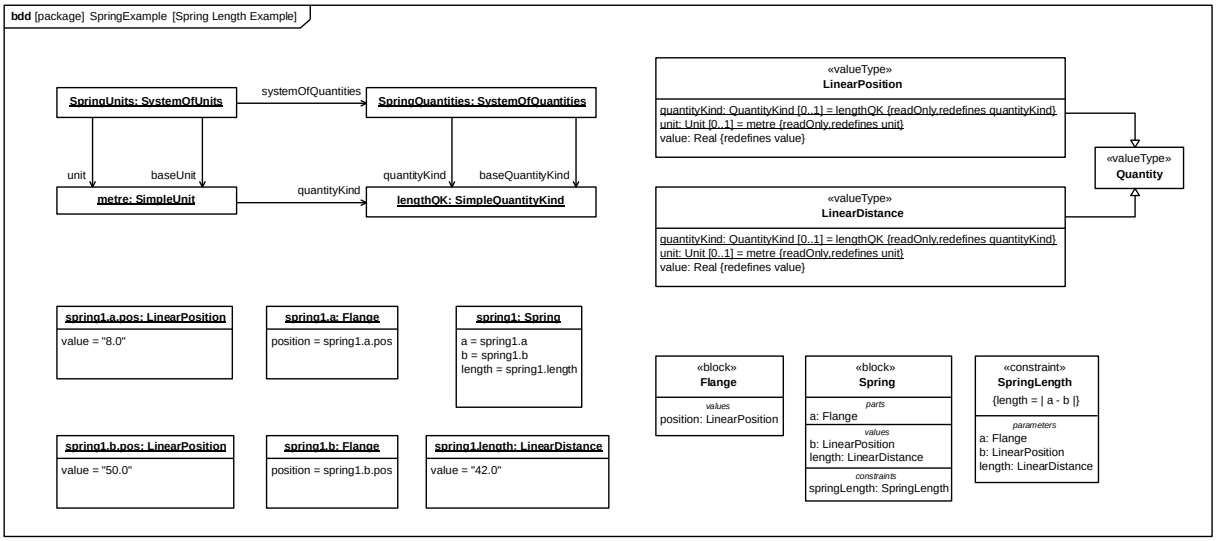

Figure C.13 Spring Length Example