Specifying Quantities in Software Models

Tanja Mayerhofer, Manuel Wimmer
Business Informatics Group, TU Wien, Austria

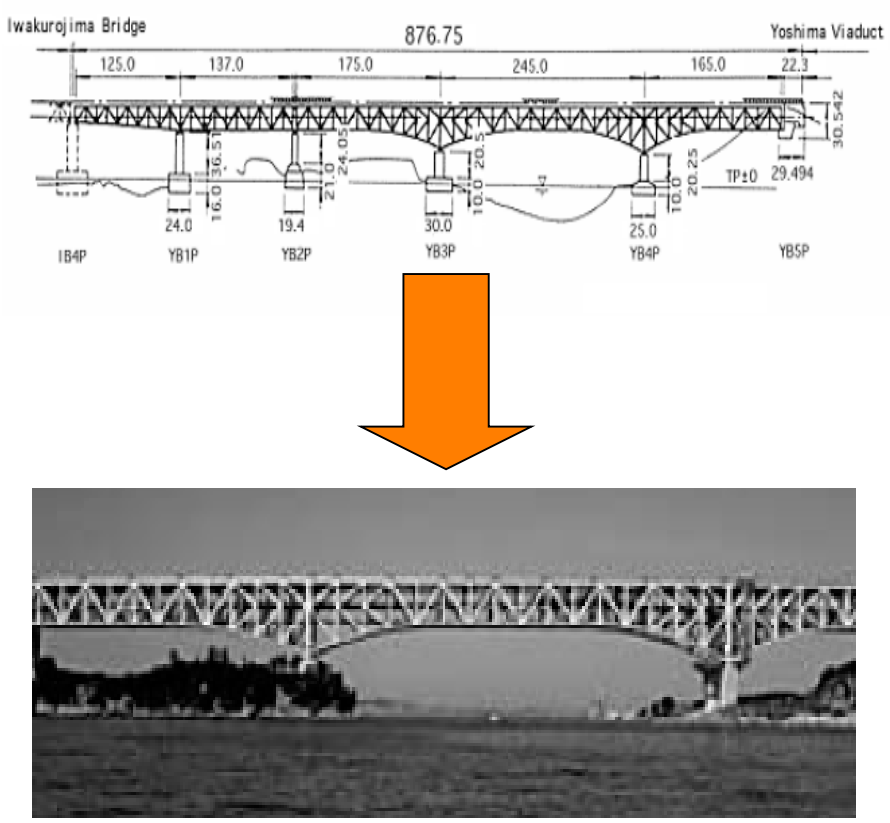
Loli Burgueño, Antonio Vallecillo
Atenea Research Group, Univ. Málaga, Spain

Universidad Complutense de Madrid

March 12, 2018

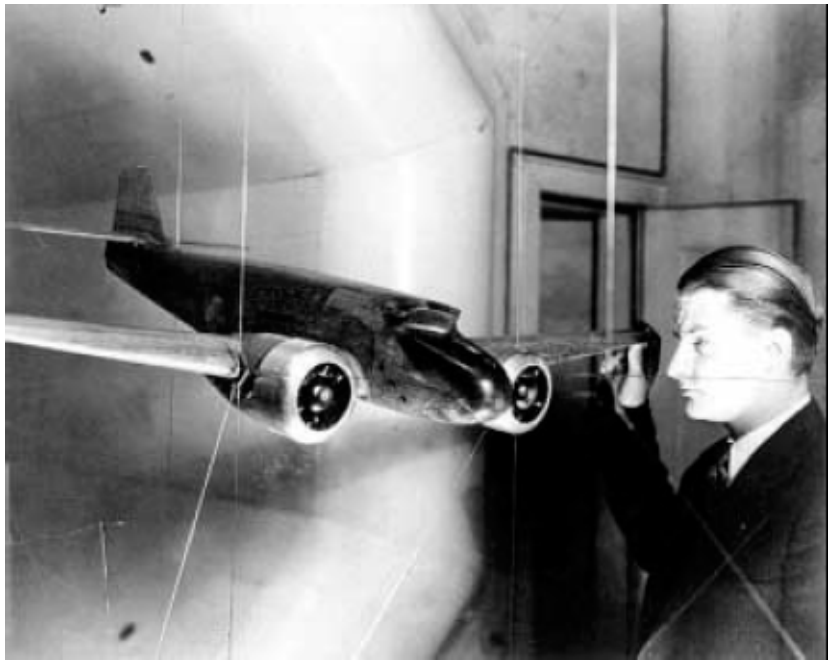
Los modelos…

- Tan antiguos como las Ingenierías (p.e. Vitruvius)
- Los ingenieros tradicionales siempre construyen modelos antes de construir sus obras y artefactos
- Los modelos sirven para:
  - **Especificar el sistema**
    - Estructura, comportamiento,…
    - Comunicarse con los distintos *stakeholders*
  - **Comprender el sistema (si ya existe)**
  - **Razonar y validar el sistema**
    - Detectar errores y omisiones en el diseño
    - Prototipado (*ejecutar* el modelo)
    - Inferir y demostrar propiedades
  - **Guiar la implementación**



Características de los modelos

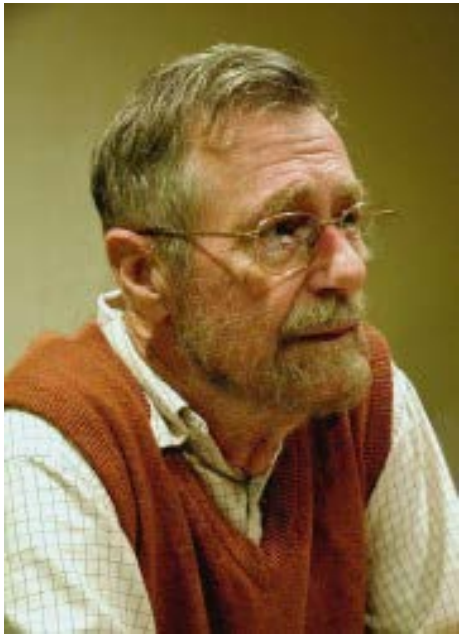
- **Abstractos**
  - Enfatizan ciertos aspectos… mientras ocultan otros
- **Comprensibles**
  - Expresados en un lenguaje comprensible por los usuarios y *stakeholders*
- **Precisos**
  - Fieles representaciones del objeto o sistema modelado
- **Predictivos**
  - Deben de poder ser usados para inferir conclusiones correctas
- **Baratos**
  - Mas fáciles y baratos de construir y estudiar que el propio sistema



Preliminaries: **Abstraction**

"Being abstract is something profoundly different from being vague... The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise."

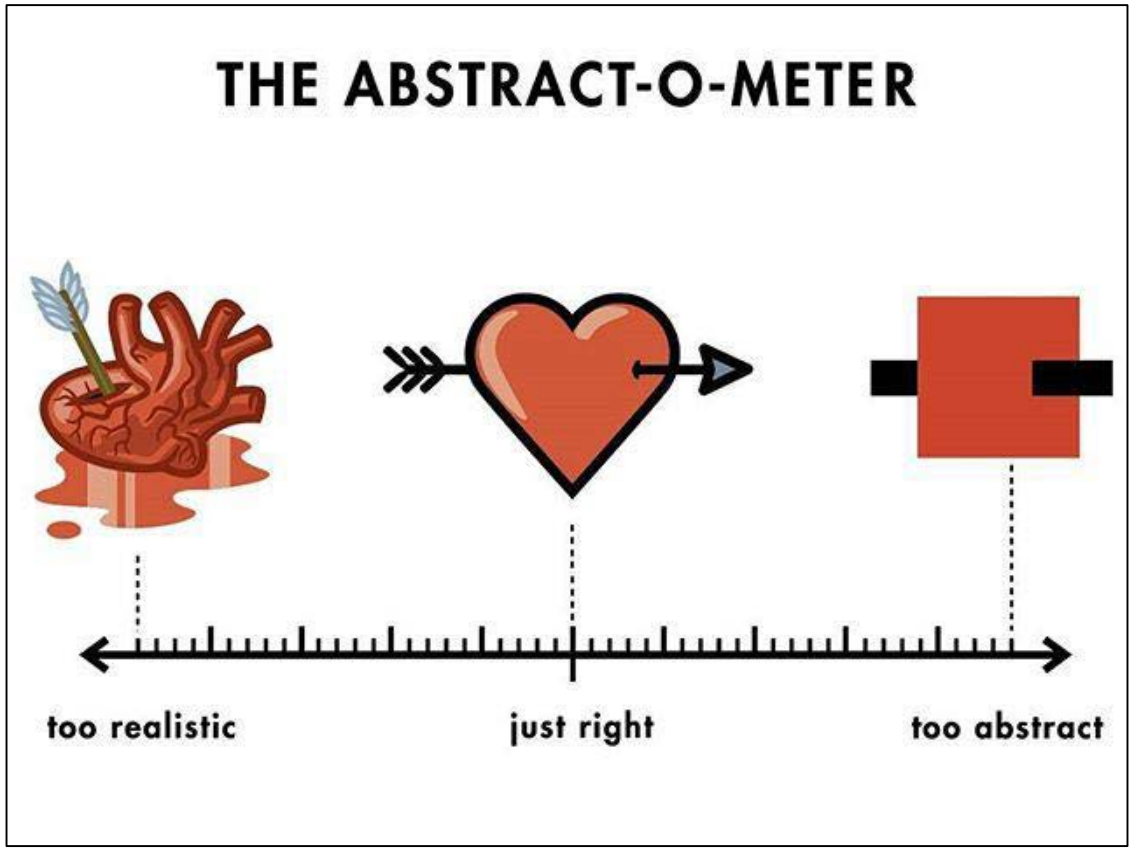
Edsger Dijkstra



Preliminaries: **The abstract-o-meter**

by Christoph Niemann (http://www.christophniemann.com)

Great depiction of one of the biggest challenges in software development: deciding the right level of abstraction for every task. Not too much. Not too little. You want to abstract irrelevant technical details (for that phase) while keeping all the key information



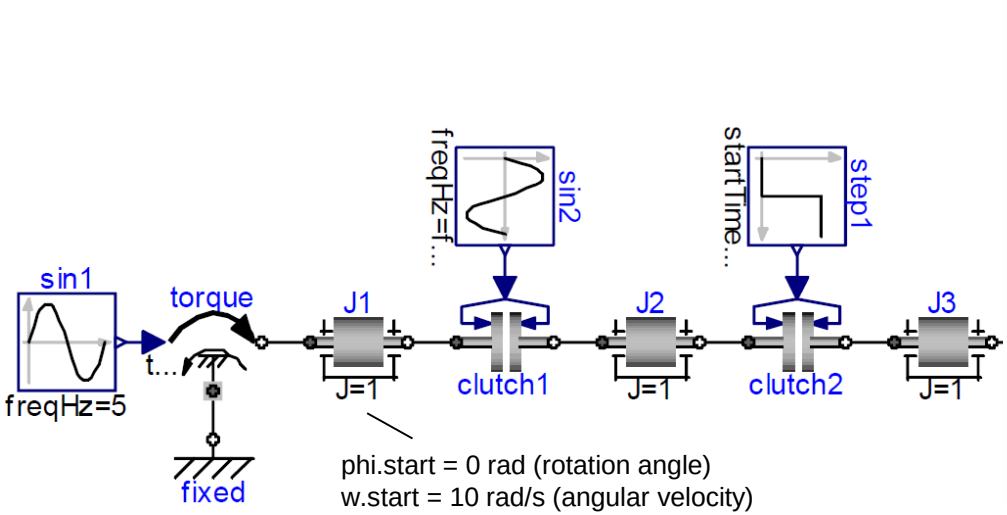
https://modeling-languages.com/essential-software-engineering-quotes-on-instagram

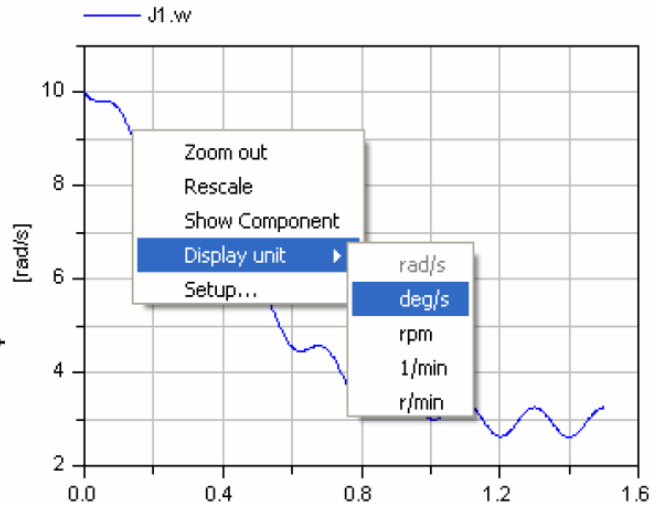
Motivation

Uncertainty and Units in Engineering Disciplines

- Engineers naturally think about *uncertainty* associated with *measured values* and *units of values*
- Uncertainty and units are explicitly defined in models and considered in model-based simulations

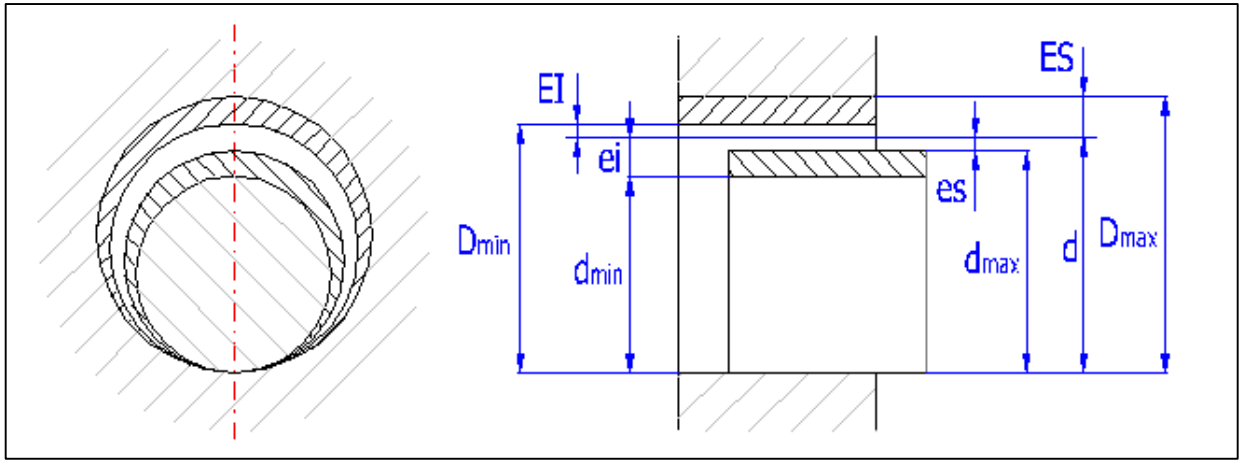
- *Example*: Coupled Clutches of Modelica Standard Library



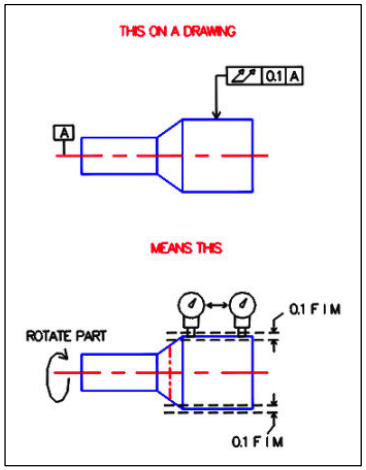


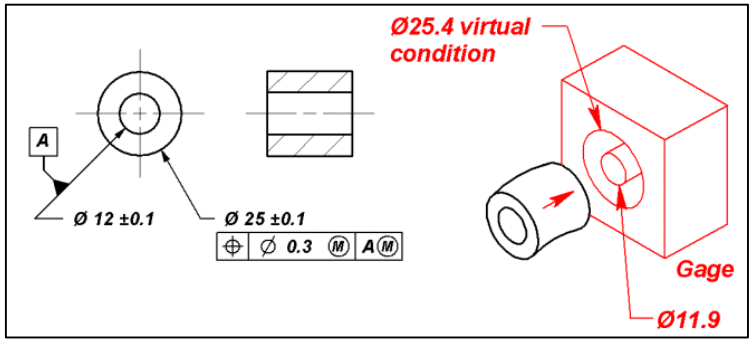
(Coupled Clutches Example of Modelica Standard Library)

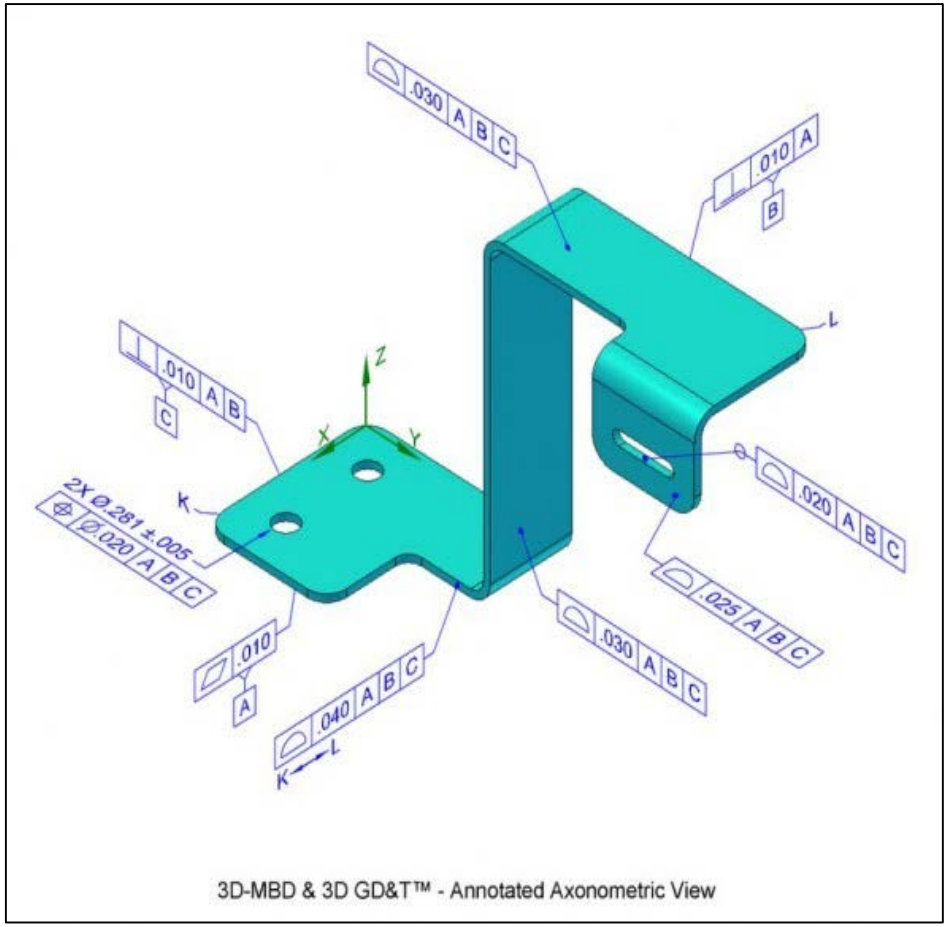
We have plenty of examples…



Units and Tolerance (Measurement Uncertainty) Examples



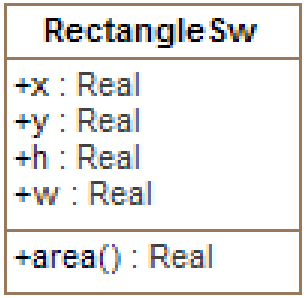




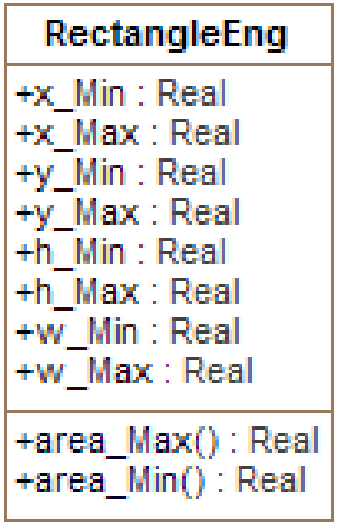
However the situation is not the same when modelled in software! ☹

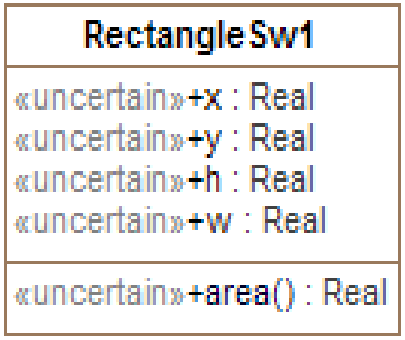
RoundObject
+posX : Real +posY : Real +posZ : Real +weight : Real +width : Real +height : Real
+move(dX : Real, dy : Real, dz : Real) +catch() +drop() +fitsIn(other : RoundObject)

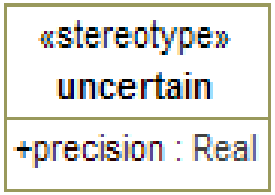
Measurement Uncertainty in Software Models – Some Attempts ☹



context RectangleSw::area() : Real = h*w





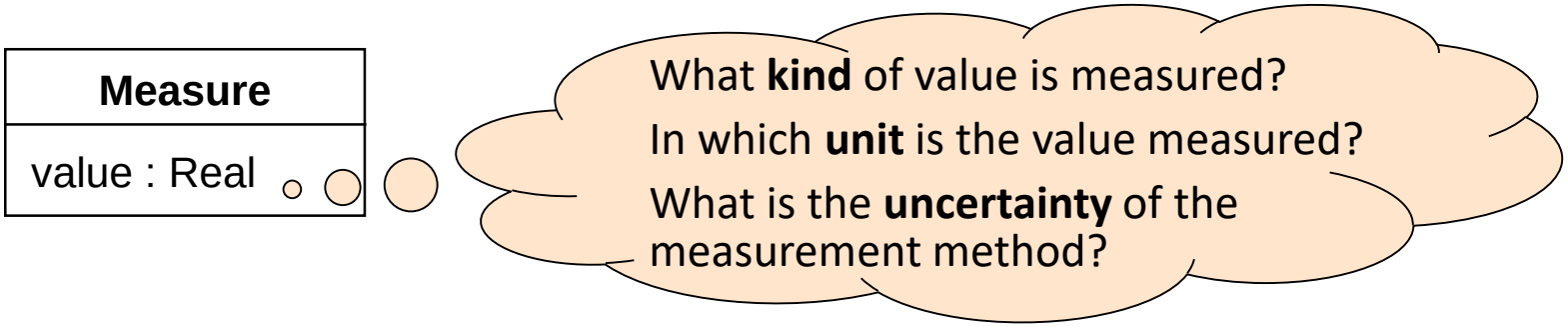


Motivation

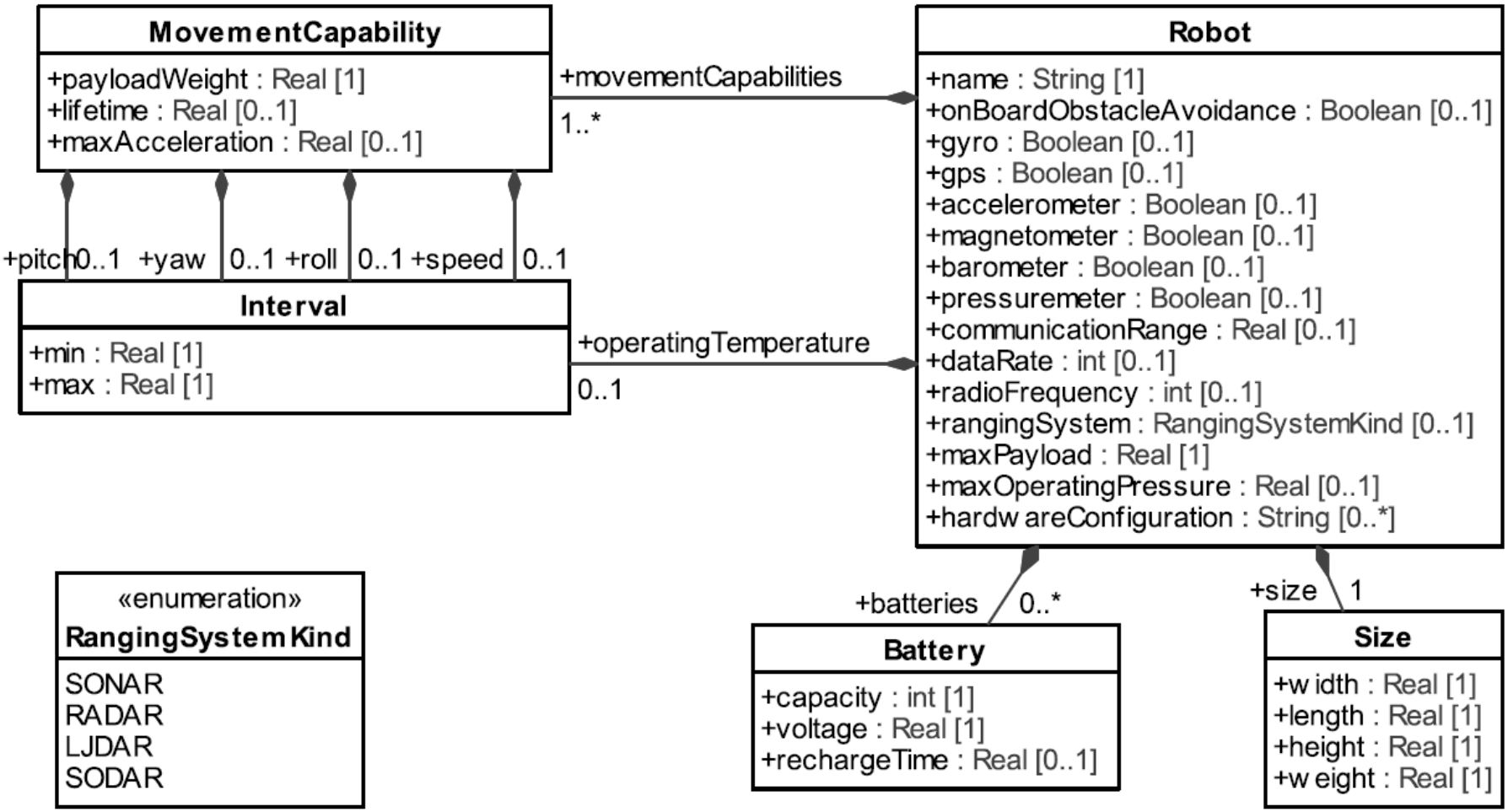
Uncertainty and Units in Software Engineering

- Very limited support for representing uncertainty and units in software models
- No support for considering such properties in model-based simulations
- Not part of their type systems!

- *Example*: How to represent a measured value in UML?

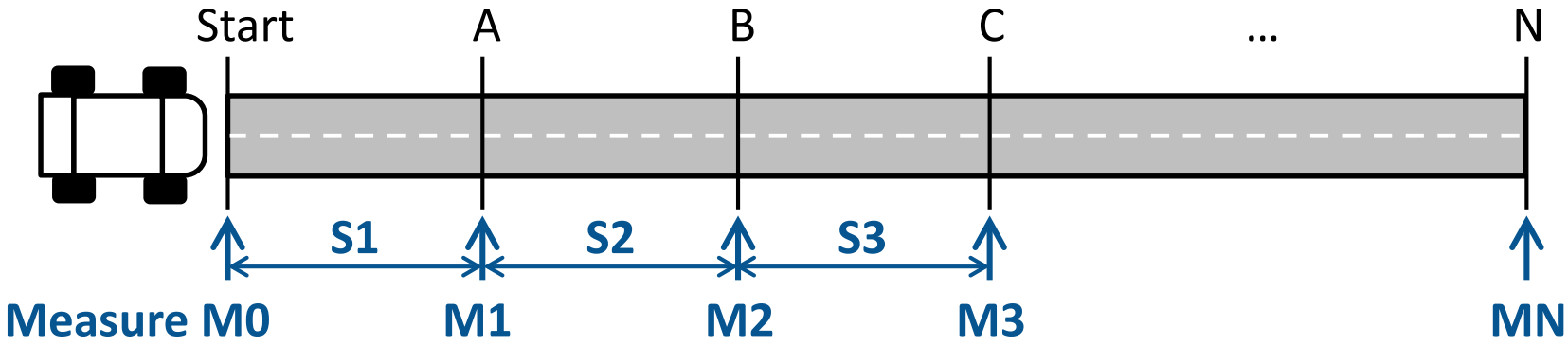


A Family of Robot Languages

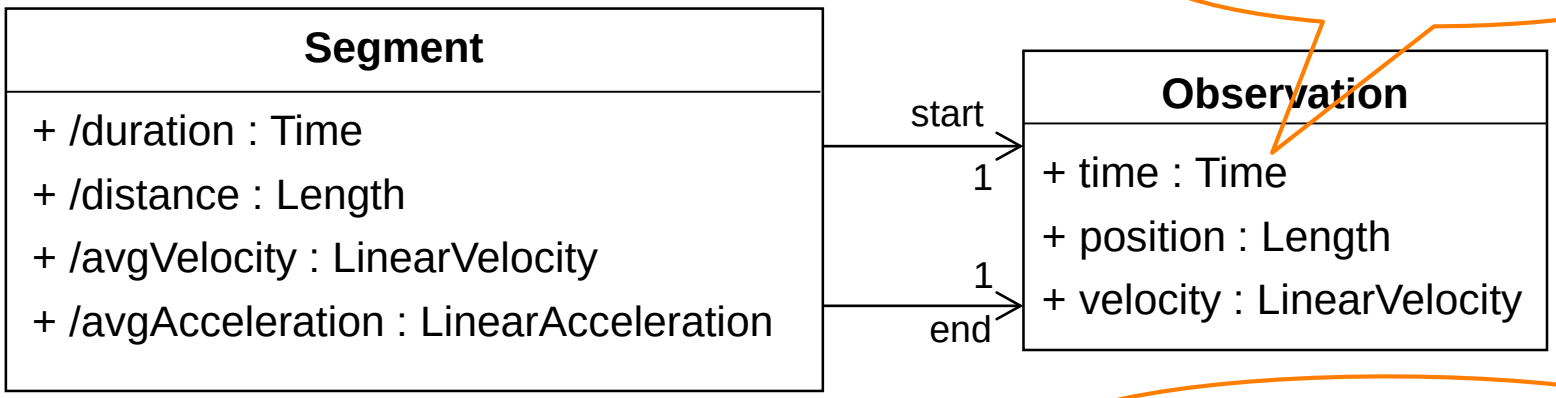


[di Ruscio et al, IEEE Access, 2017]

Example: The way we would like to model with units and uncertain data



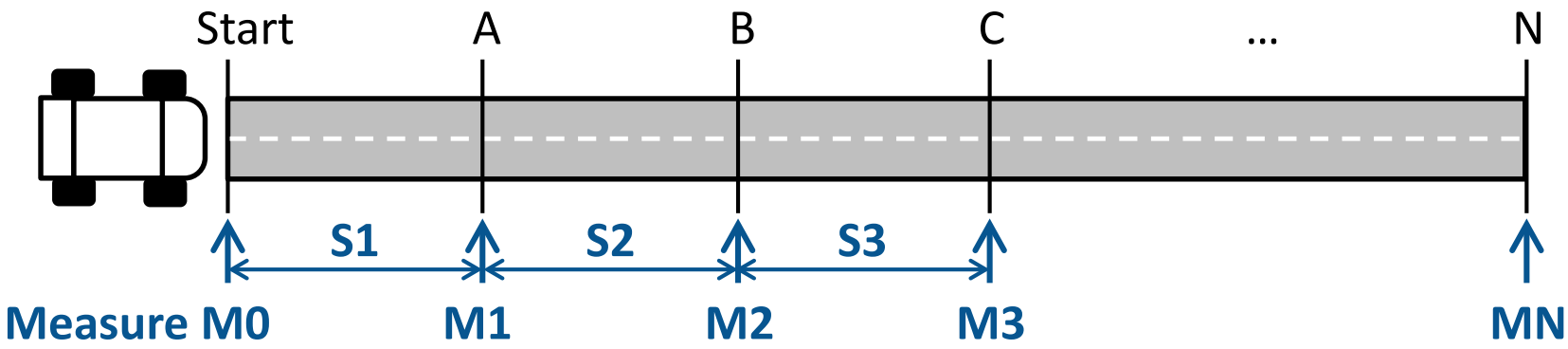
Values incorporate uncertainty and units!



Operations are type-safe and handle uncertainty!

```
duration = end.time.minus(start.time)
distance = end.position.minus(start.position)
avgVelocity = distance.divideBy(duration)
avgAcceleration =(end.velocity.minus(start.velocity)).divideBy(duration)
```

Example: The way we would like to model with units and uncertain data



S1 : Segment

/duration = 10.0 ± **0.0019799 s**
/distance = 10.0 ± **0.0014142 m**
/avgVelocity = 1.0000000392 ± **0.000489 m/s**
/avgAcceleration = 0.200000008 ± **0.0632468 m/s²**

start →

M0 : Observation

time = 0.0 ± **0.0014 s**
position = 0.0 ± **0.001 m**
velocity = 0.0 **m/s**

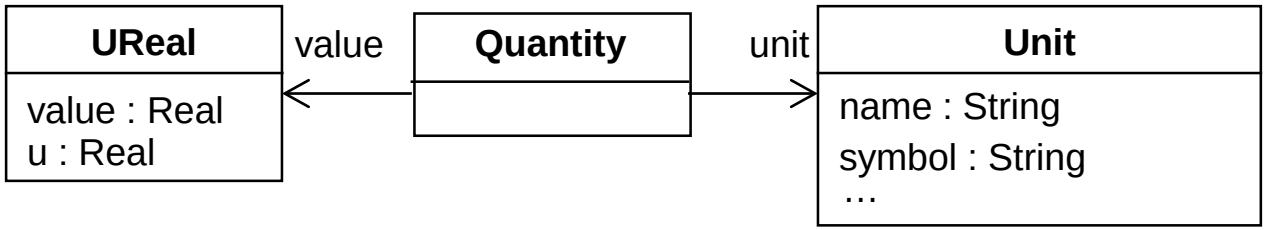
end →

M1 : Observation

time = 10.0 ± **0.0014 s**
position = 10.0 ± **0.001 m**
velocity = 2.0 ± **0.02 m/s**

Quantities

- A Quantity is an observable property of an object, event or system that can be measured and quantified numerically. [QUDT]
  - For example its Mass, Speed or Temperature
- Quantities are determined by two main attributes, ***kind*** and ***magnitude***, which are expressed by a ***numerical value*** and a ***unit of measure***. [NIST SI]
  - For example, 3.5 m/s
- The numerical value should also incorporate a statement of the associated uncertainty [GUM]:
  - "When dealing with real-world entities, models need to take into account the inability to know, estimate or measure with complete precision the value of the quantity… A measurement result can only be considered complete when it is accompanied by a statement of the associated uncertainty" [GUM]



International System of Units (SI, ISO 80000)

- Base dimensions: Length, Mass, Time, Electric Current, Thermodynamic Temperature, Amount of Substance, Luminous Intensity, Data Storage Capacity, Entropy, Traffic Intensity, Level, Angle
- Base units: Meter (m), Kilogram (kg), Second (s), Ampere (A), Kelvin (K), Mole (mol), Candela (cd), Bit (b), Shannon (Sh), Erlang (E), Decibel (dB), Radian (rad)
- Derived dimensions: 100+ dimensions derived from the base dimensions e.g., Area, Volume, LinearVelocity.
- Derived units: 100+ units derived from the base units e.g., Square Meter (m^2), Cubic Meter (m^3), Meter per Second (m/s)

Other Systems of Units

- Centimeter-Gram-Second System (CGS)
- Imperial System
- United States Customary System (USCS, USC)

B. N. Taylor and A. Thompson. The International System of Units (SI). NIST, 2008. http://www.nist.gov/pml/pubs/sp811/.

Uncertainty

1. **Uncertainty**: Quality or state that involves imperfect and/or unknown information. It applies to predictions of future events, estimations, physical measurements, or unknown properties of a system, due to:
   - Underspecification
   - Lack of knowledge of the system actual behavior or underlying physics
   - *Numerical or measurement errors*
   - *Numerical approximations because values are too costly to measure*
   - *Associated properties are not directly measurable or accessible*
2. **Measurement of Uncertainty**: A set of possible states or outcomes where probabilities are assigned to each possible state or outcome.
3. The ISO document "**Guide to the Expression of Uncertainty in Measurement**" (GUM) describes the procedure for calculating measurement uncertainty, as used by most Engineering Disciplines (*but Software, until very recently*)

[GUM] JCGM 100:2008. Evaluation of measurement data – Guide to the expression of uncertainty in measurement. http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf

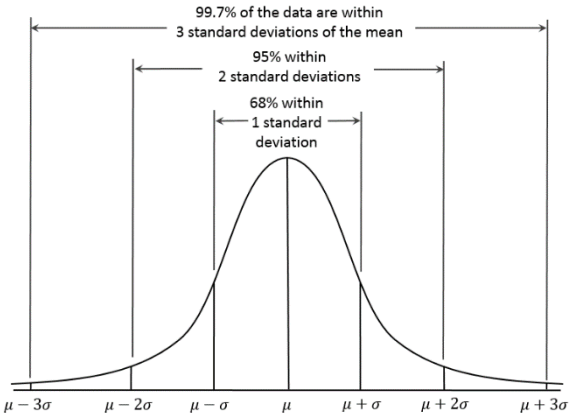
Definition: Standard Uncertainty [GUM]

- Uncertainty of the result of a measurement x expressed as a standard deviation u
- Representation: $x \pm u$ or (x, u)
- *Examples:*

  Normal distribution: (x, σ) with mean x, standard deviation σ

  Interval $[a, b]$: Uniform or rectangular distribution is assumed

$$(x, u) \text{ with } x = \frac{a+b}{2}, u = \frac{(b-a)}{2\sqrt{3}}$$



[GUM] JCGM 100:2008. Evaluation of measurement data – Guide to the expression of uncertainty in measurement. http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf

- Computations with uncertain values have to respect the *propagation of uncertainty* (uncertainty analysis)

Two Methods for Computing Aggregated Uncertainty

- Normal or Uniform distribution: Analytical (closed-form) solutions
- General case: Using samples (SIPMath Std.)

A. Vallecillo, C. Morcillo, and P. Orue. Expressing Measurement Uncertainty in Software Models. In Proc. of QUATIC 2016, pages 1–10, 2016.

Assumptions / Facts

- **Uncertainty** needs to be part of any value representing a physical property
- **Units** should be an intrinsic part of any measured or calculated value
- **Dimensions** should act as types (not “units as types”)
  - Same as when you add two “integers” — no matter if they are expressed in Octal, Hex or Decimal bases
  - For example, consider the following Java program:

```
int x, y, sum;
x = 0x13;  y = 10;   sum = x + y;
System.out.format(“The sum of %d and %d is %d (in decimal) and
                                %x (in hexa).%n", x, y, sum, sum);
```

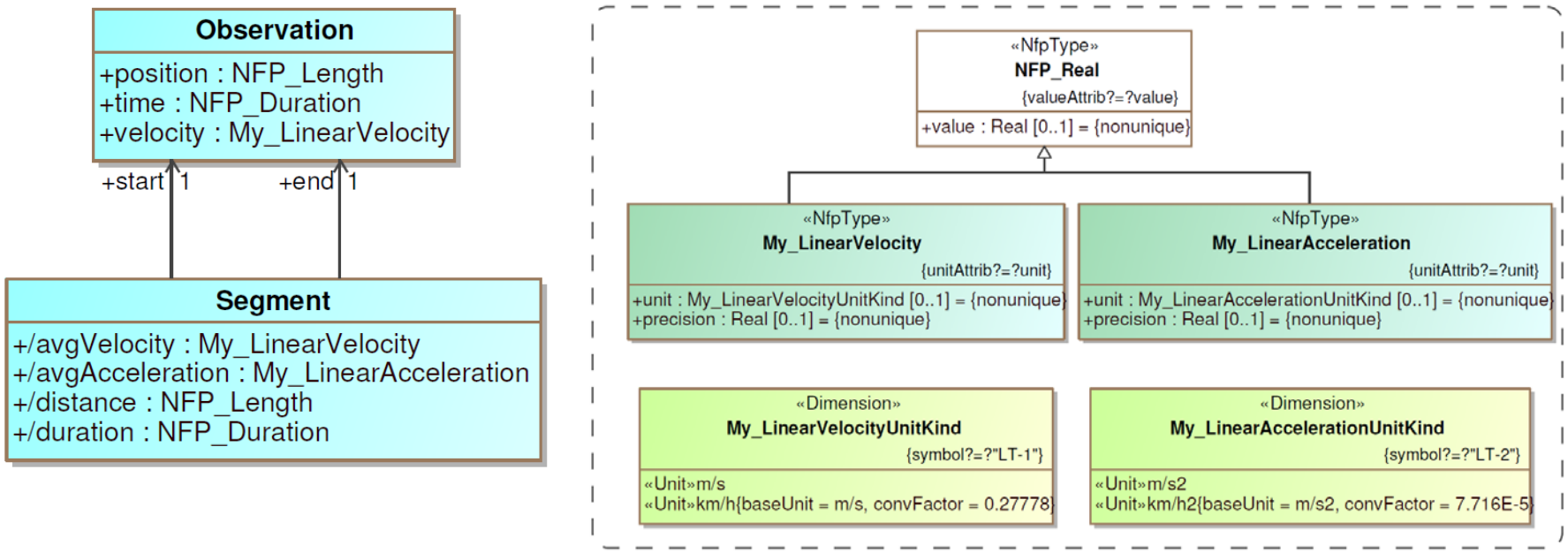
Result: “The sum of 19 and 10 is 29 (in decimal) and 1d (in hexa).”

  - The key are the dimensions, not the units in which values are expressed
  - The unit-mismatch problem is no longer an issue (units always accompany values)
- The type system should ensure **type-safe operations**
  - Not permitting illegal or invalid operations on dimensions (e.g. Time+Length)

Quantities in UML Models (MARTE and SysML)

- MARTE has stereotypes to decorate values with information about the units they are expressed in, and with measurement uncertainty ("precision")
  - However:
    - It is simply "decorative information": no type checking, no operations for aggregating uncertainty values
    - Units used as types! (not "Dimensions as types")
- SysML 1.4 provides the QUDV (Quantities, Units, Dimensions) and ISO 80000 library with all units and dimensions.
  - However:
    - No support for dealing with measurement uncertainty
    - Types of values are still based on explicit units
- *Compatibility problems when combining MARTE and SysML models*
- NOTE: Of course, simulation tools (Modelica, Matlab/Simulink) and mathematic languages (Mathematica) provide support for units, dimensions and uncertainty, but they are at a different abstraction level

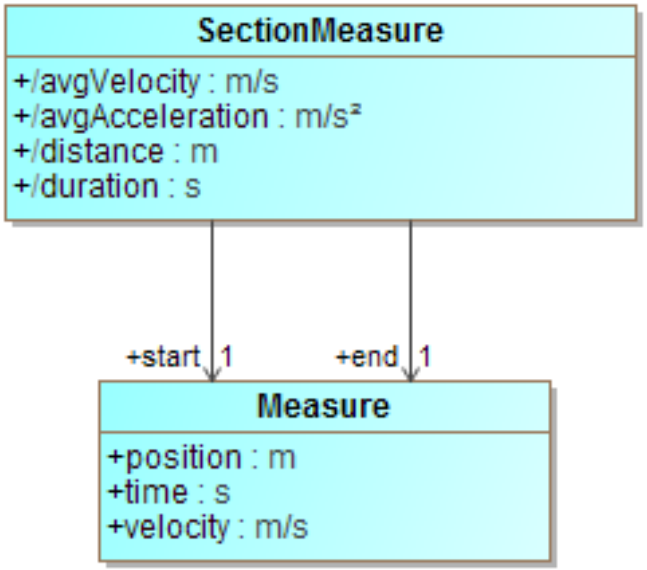
MARTE Specification of the Example



- We need to define new (proprietary!) dimensions and new units because they are not provided by the MARTE library of basic types
- It is simply “decorative information”: New types and units do not have methods, there are no operations for aggregating uncertainty values, no static type-checking, no standard derivation rules for operations on values of these types, no propagation of uncertainty
- Cumbersome and error prone process

SysML Specification of the Example (1/2)

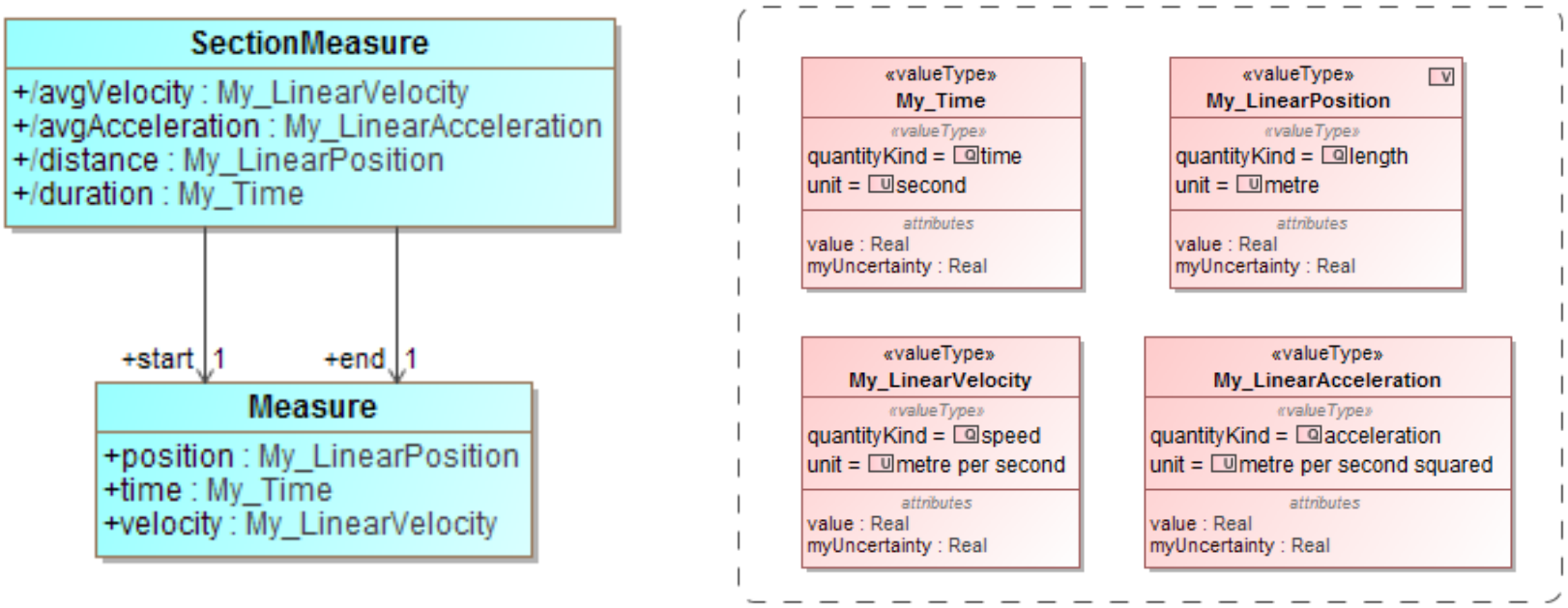
[SysML v1,4, Sect 8.4.1, Fig 8.12]



- This is the simplest way of dealing with units in SysML (using the library "SI ValueType Library")
- Types of values are based on explicit units…
- No way of handling uncertainty
- It is simply "decorative information": Values do not have methods, no static type-checking, no derivation rules for operations on values of these types

SysML Specification of the Example (2/2)

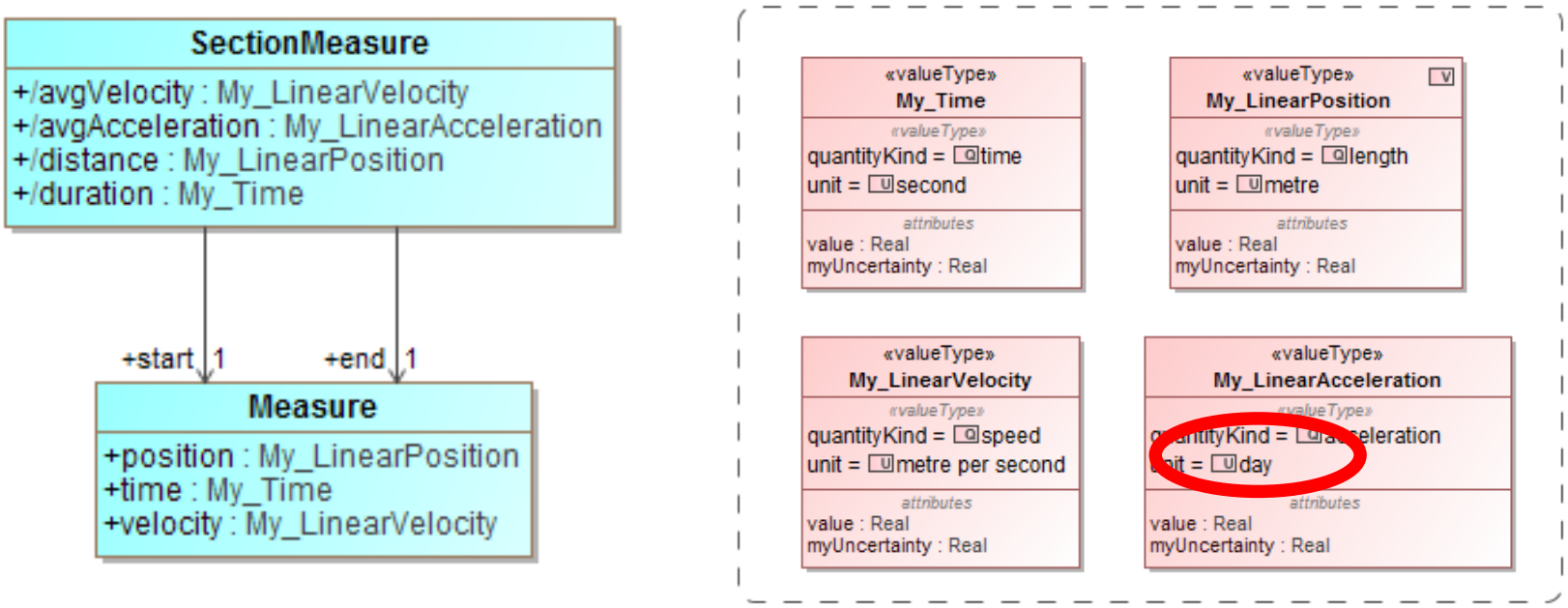
[SysML v1,4, Sect E.5.4.2, Fig. E.9]



- Although this approach uses dimensions, types of values are still based on units
- Users have to define their own Quantities (“value” and “myUncertainty”). No standard and compatible way of handling uncertainty
- It is simply “decorative information”: New types and units do not have methods, there are no operations for aggregating uncertainty values, no static type-checking, no derivation rules for operations on values of these types

SysML Specification of the Example (2/2)

[SysML v1,4, Sect E.5.4.2, Fig. E.9]



- Although this approach uses dimensions, types of values are still based on units
- Users have to define their own Quantities (“value” and “myUncertainty”). No standard and compatible way of handling uncertainty
- It is simply “decorative information”: New types and units do not have methods, there are no operations for aggregating uncertainty values, no static type-checking, no derivation rules for operations on values of these types
- **High chances of mistakes in the definitions!!!**

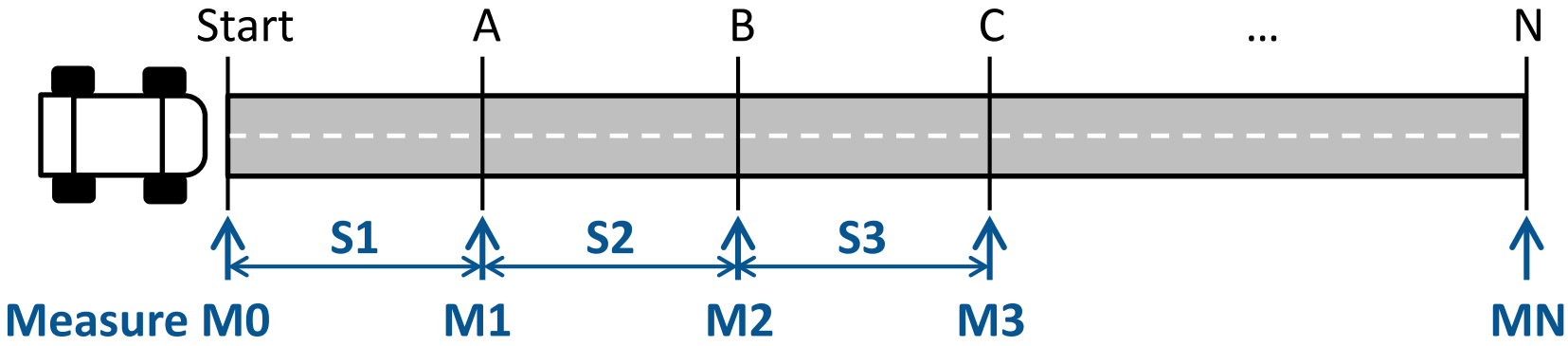
Our Work

1. **Type system** for representing measurement uncertainty and units
   - Kernel representation for Quantities (value + uncertainty)
   - All SI *Dimensions* and their operations available (Length, Area, Volume, Force, Torque, etc.) as specialization of “Quantity”
   - All SI Units available

2. **Algebra of operations** for performing computations with units and uncertain data
   - Computational kernel for computing quantities
   - Type-safe operations
     - Length + Length returns a Length
       Length / Time returns a LinearVelocity
       Length + Time is not valid
     - *Independently from whether the Length is expressed in meters, feet or inches*
   - Propagation of uncertainty

3. **Implementations** in UML, OCL, fUML y Java

Example: The way we would like to model with units and uncertain data



Values incorporate uncertainty and units!

SectionMeasure

+ /duration : Time
+ /distance : Length
+ /avgVelocity : LinearVelocity
+ /avgAcceleration : LinearAcceleration

start 1

1 end

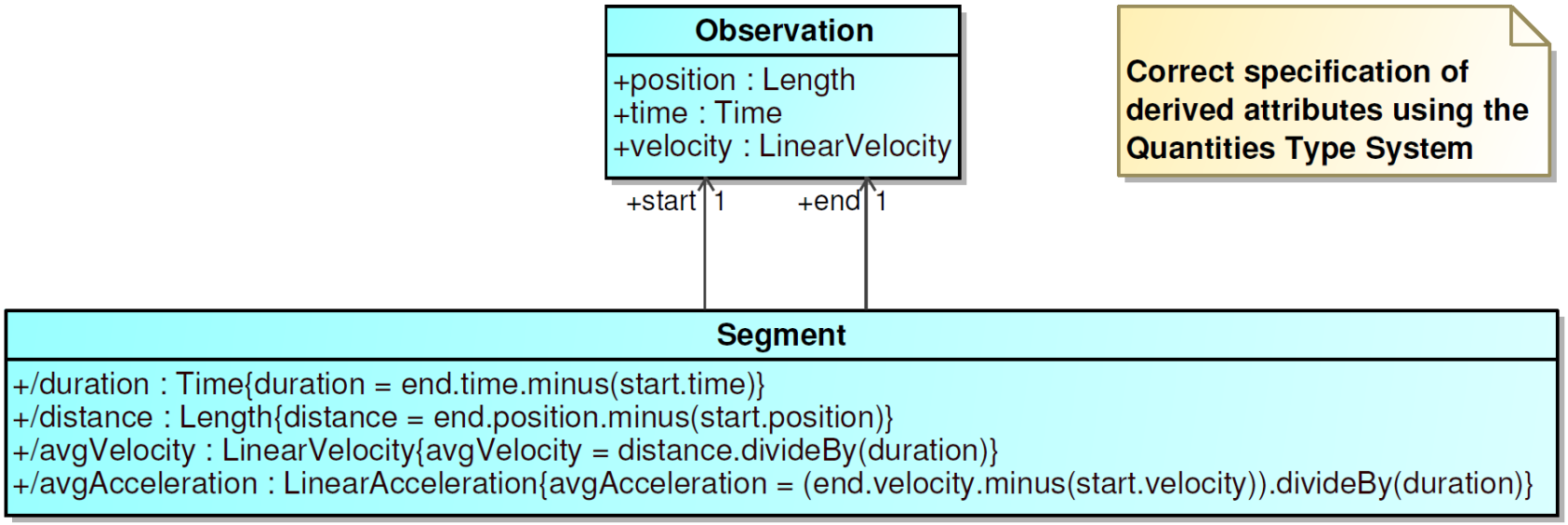
Measure

+ time : Time
+ position : Length
+ velocity : LinearVelocity

Operations are type-safe and handle uncertainty!

```
duration = end.time.minus(start.time)
distance = end.position.minus(start.position)
avgVelocity = distance.divideBy(duration)
avgAcceleration =(end.velocity.minus(start.velocity)).divideBy(duration)
```

Modeling the Example with our Proposal



- *Quantities* as types for values (providing both unit and uncertainty info).
- Beyond "decorative" information:
  - Quantities provide methods for aggregating uncertainty values
  - Standard derivation rules for operations on values of these types
  - Static type-checking of operations with quantities and automatic conversion of units (no unit-mismatch problem)

Example of Available Quantities (Dimensions)

Length
Length() Length(u : Real) Length(u : UReal, unit : Unit) Length(q : Quantity) Length(x : Real) Length(x : Real, u : Real) Length(x : Real, unit . Unit) Length(x : Real, u : Real, unit : Unit) add(r : Length) : Length minus(r : Length) : Length mult(r : Mass) : LengthMass multr(: Thermodynamic Temperature) : LengthTemperature mult(r : Length) : Area mult(r : Area) : Volume mult(r : MassPerUnitTime) : LinearMomentum divideBy(r : Time) : LinearVelocity divideBy(r : ThermodynamicTemperature) : LinearThermalExpansion abs() : Length neg() : Length equals(r : Length) : Boolean distinct(r : Length) : Boolean floor() : Length round() : Length min(r : Length) : Length max(r : Length) : Length

LinearVelocity
LinearVelocity() LinearVelocity(u : Real) LinearVelocity(u : UReal, unit : Unit) LinearVelocity(q : Quantity) LinearVelocity(x : Real) LinearVelocity(x : Real, u : Real) LinearVelocity(x : Real, unit . Unit) LinearVelocity(x : Real, u : Real, unit : Unit) add(r : LinearVelocity) : LinearVelocity minus(r : LinearVelocity) : LinearVelocity mult(r : MassPerUnitTime) : Force mult(r : Mass) : LinearMomentum mult(r : Time) : Length divideBy(r : Time) : LinearAcceleration abs() : LinearVelocity neg() : LinearVelocity equals(r : LinearVelocity) : Boolean distinct(r : LinearVelocity) : Boolean floor() : LinearVelocity round() : LinearVelocity min(r : LinearVelocity) : LinearVelocity max(r : LinearVelocity) : LinearVelocity

How is this implemented?

- Any unit can be derived from the base units:
  $B_1^{e_1} * B_2^{e_2} \ldots B_n^{e_n}$ where B_i represents a base unit and e_i its exponent
- Hence, any unit can be defined by the exponents e_i of the base units:
  $\langle e_1, e_1, \ldots, e_n \rangle$
- *Examples*

$$Meter\ (m) = m^1 * kg^0 * s^0 * A^0 * K^0 * cd^0 * mol^0 * rad^0 = \langle 1,0,0,0,0,0,0,0 \rangle$$

$$Square\ Meter\ (m^2) = m^2 * kg^0 * s^0 * A^0 * K^0 * cd^0 * mol^0 * rad^0 = \langle 2,0,0,0,0,0,0,0 \rangle$$

$$Meter\ per\ Second\ (m/s) = m^1 * kg^0 * s^{-1} * A^0 * K^0 * cd^0 * mol^0 * rad^0 = \langle 1,0,-1,0,0,0,0,0 \rangle$$

R. Hodgson, P. J. Keller, J. Hodges, and J. Spivak. QUDT – Quantities, Units, Dimensions and Data Types Ontologies. TopQuadrant, Inc. and NASA AMES Research Center, 2014. http://qudt.org/.

- Conversion of quantity values from base units B_i to derived units D_i
  - Multiply the numerical value of the quantity value with conversion factor cf
  - Add an offset o to the resulting numerical value
- Definition: $x\ D_i\ = (x\ * cf_i\ +\ o_i)\ B_i$
- *Examples*:

$$x\ km = (x\ * 1000\ +\ 0)\ m$$

$$x\ °C = (x\ * 1 + 273.15)\ \mathrm{K}$$

$$x\ km/h = (x\ * \frac{1000}{3600} + 0)\ \mathrm{m/s}$$

- Conversion factors and offsets can be defined relative to the base units: $cf_i: \langle cf_1, cf_1, \dots, cf_n \rangle$, $o_i: \langle o_1, o_2, \dots, o_n \rangle$
- *Examples*:

$$Kilometer\ (km):\ cf = \langle 1000, 1, 1, 1, 1, 1, 1, 1 \rangle,\quad of = \langle 0, 0, 0, 0, 0, 0, 0, 0 \rangle$$

$$Celcius\ (°C):\ cf = \langle 1, 1, 1, 1, 1, 1, 1, 1 \rangle,\quad of = \langle 0, 0, 0, 0, 273.15, 0, 0, 0 \rangle$$

$$Kilometer\ per\ Hour\ (km/h):\ cf = \langle 1000, 1, 3600, 1, 1, 1, 1, 1 \rangle,\ of = \langle 0, 0, 0, 0, 0, 0, 0, 0 \rangle$$

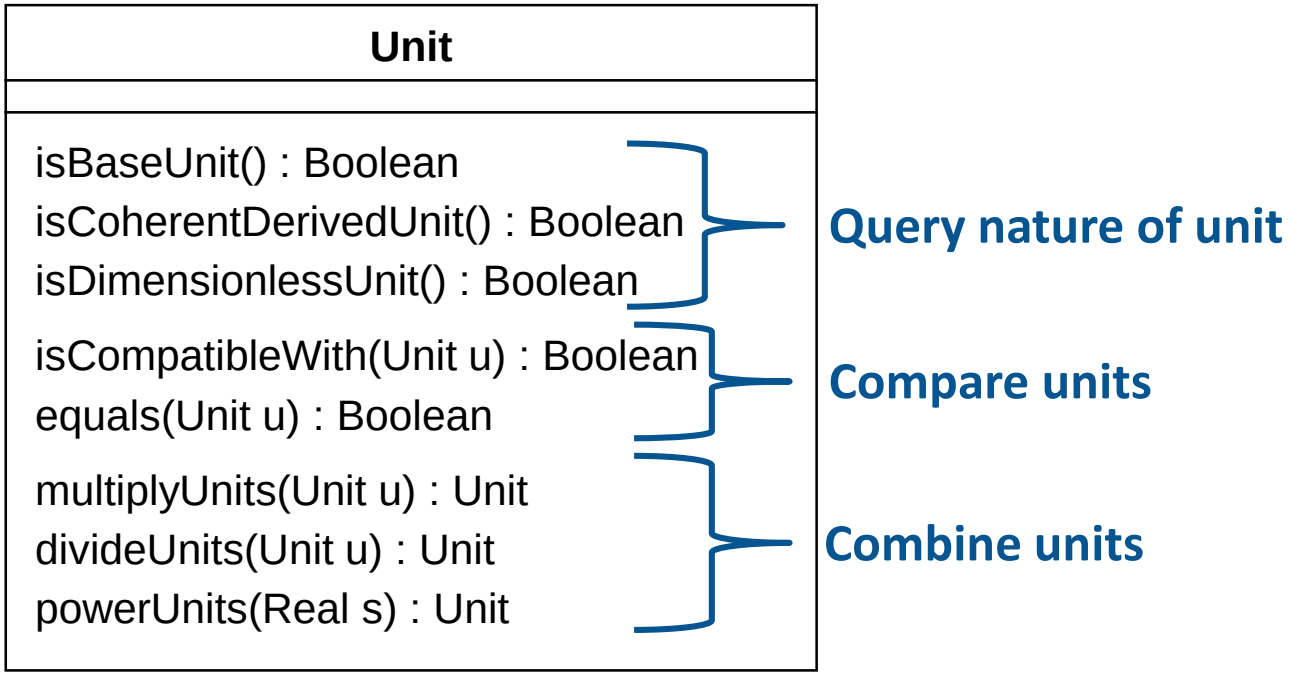
- Domain Model

Unit
name : String symbol : String dimensions : Real [12] conversionFactor : Real [12] offset : Real [12]

- *Example Instances*

m : Unit
name = "Meter" symbol = "m" dimensions = <1,0,0,0,0,0,0,0,0,0,0,0,0> conversionFactor = <1,1,1,1,1,1,1,1,1,1,1,1,1> offset = <0,0,0,0,0,0,0,0,0,0,0,0,0>

km/h : Unit
name = "Kilometer per Hour" symbol = "km/h" dimensions = <1,0,-1,0,0,0,0,0,0,0,0,0,0> conversionFactor = <1000,1,3600,1,1,1,1,1,1,1,1,1,1> offset = <0,0,0,0,0,0,0,0,0,0,0,0,0>


Unit
isBaseUnit() : Boolean
isCoherentDerivedUnit() : Boolean
isDimensionlessUnit() : Boolean
isCompatibleWith(Unit u) : Boolean
equals(Unit u) : Boolean
multiplyUnits(Unit u) : Unit
divideUnits(Unit u) : Unit
powerUnits(Real s) : Unit
Query nature of unit
Compare units
Combine units

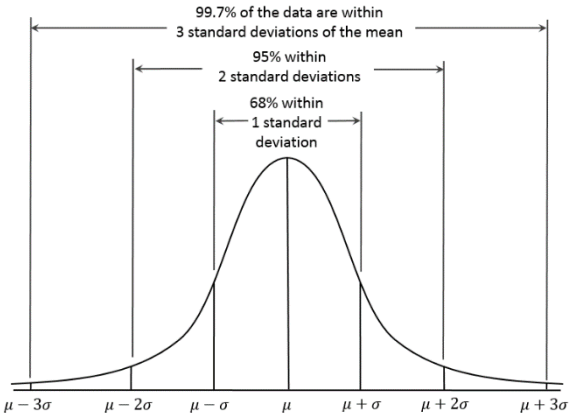
Definition: Standard Uncertainty [GUM]

- Uncertainty of the result of a measurement x expressed as a standard deviation u
- Representation: $x \pm u$ or (x, u)
- *Examples:*

  Normal distribution: (x, σ) with mean x, standard deviation σ

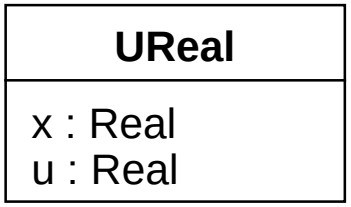
  Interval $[a, b]$: Uniform or rectangular distribution is assumed

$$(x, u) \text{ with } x = \frac{a+b}{2}, u = \frac{(b-a)}{2\sqrt{3}}$$



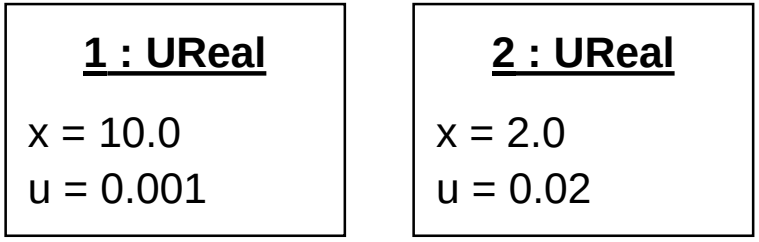
[GUM] JCGM 100:2008. Evaluation of measurement data – Guide to the expression of uncertainty in measurement. http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf

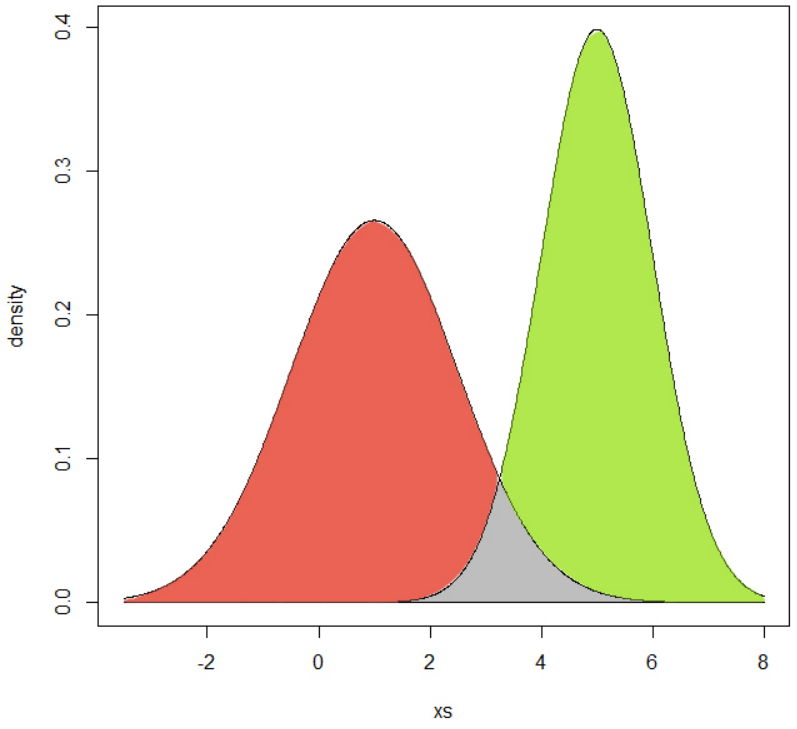
- Domain Model



- *Example Instances*

10 ± 0.001 2 ± 0.02





a: (1.0,1.5) → μ_a=1; σ_a=1.5

b: (5.0,1.0) → μ_b=5; σ_b=1

Comparison results

a = b: false

a < b: true

a > b: false

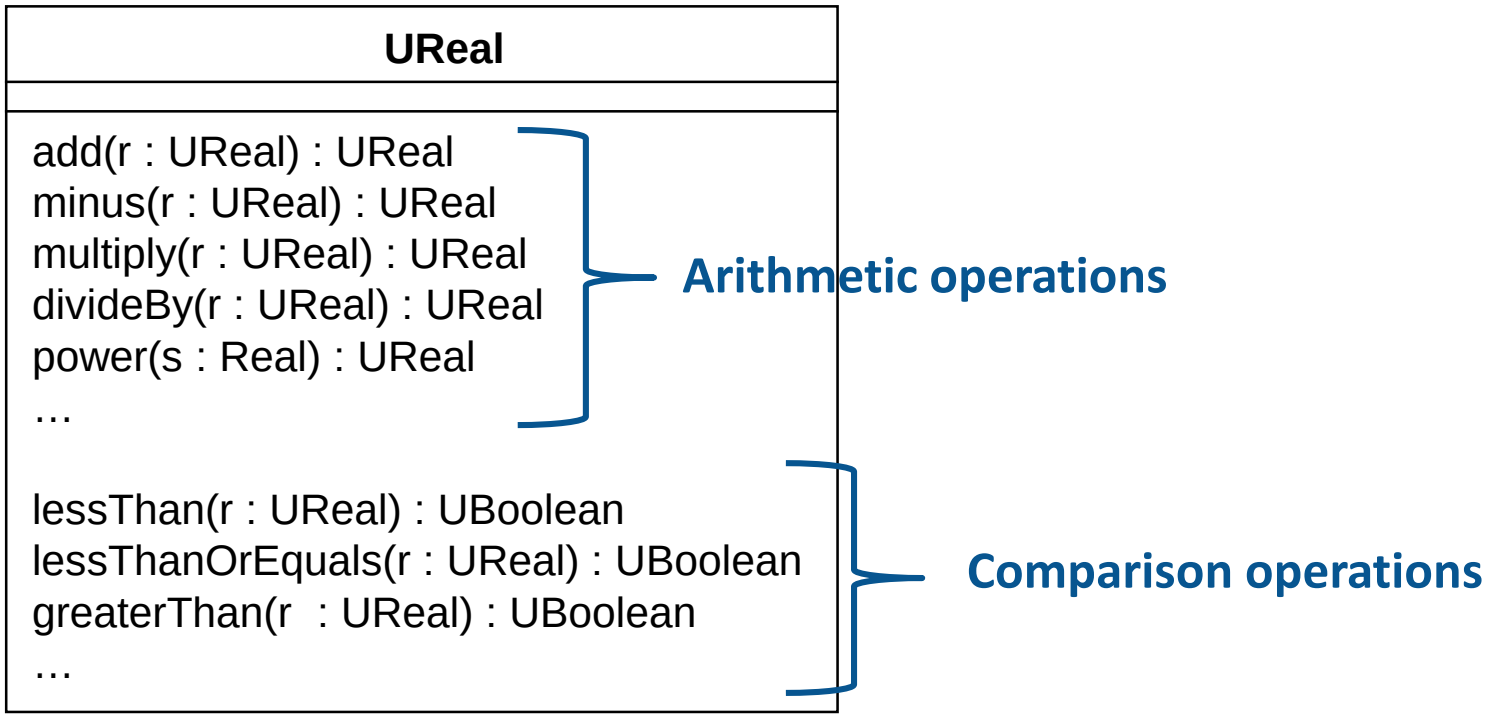
p(a = b): 0.1068663258850953 (gray area)

p(a < b): 0.8931336674114904 (red area)

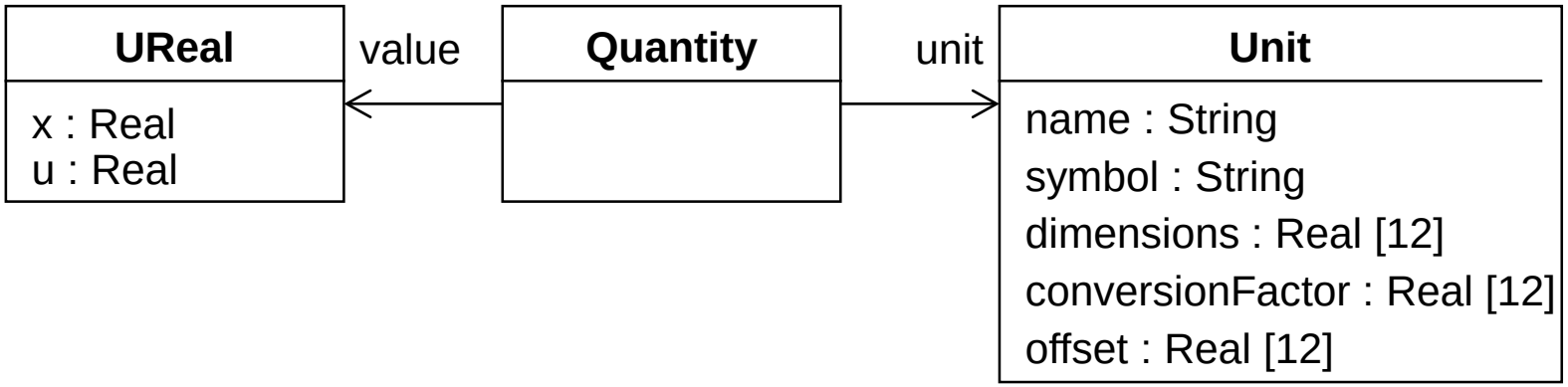
p(a > b): 1.1102230246251565E-16

p(a <= b): 0.9999999999999999 (“<“ + “=“)

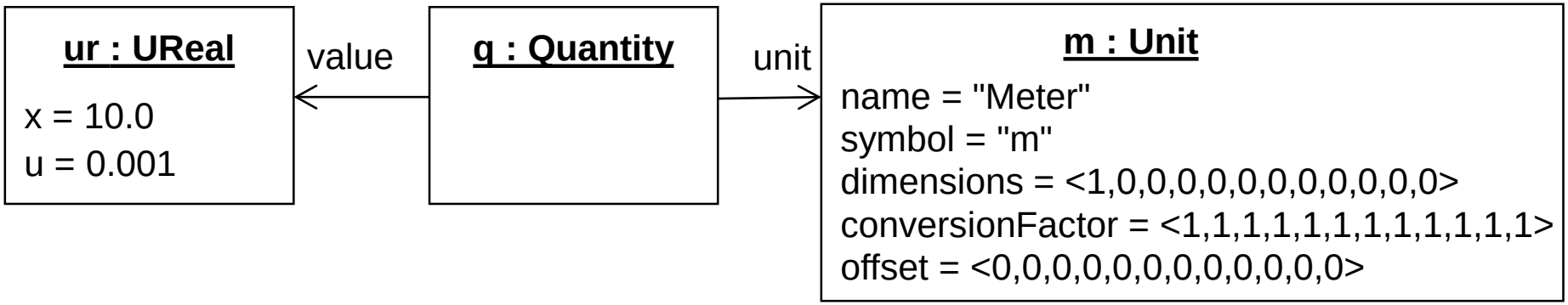
p(a >= b): 0.1068663258850964 (“>” + “=“)

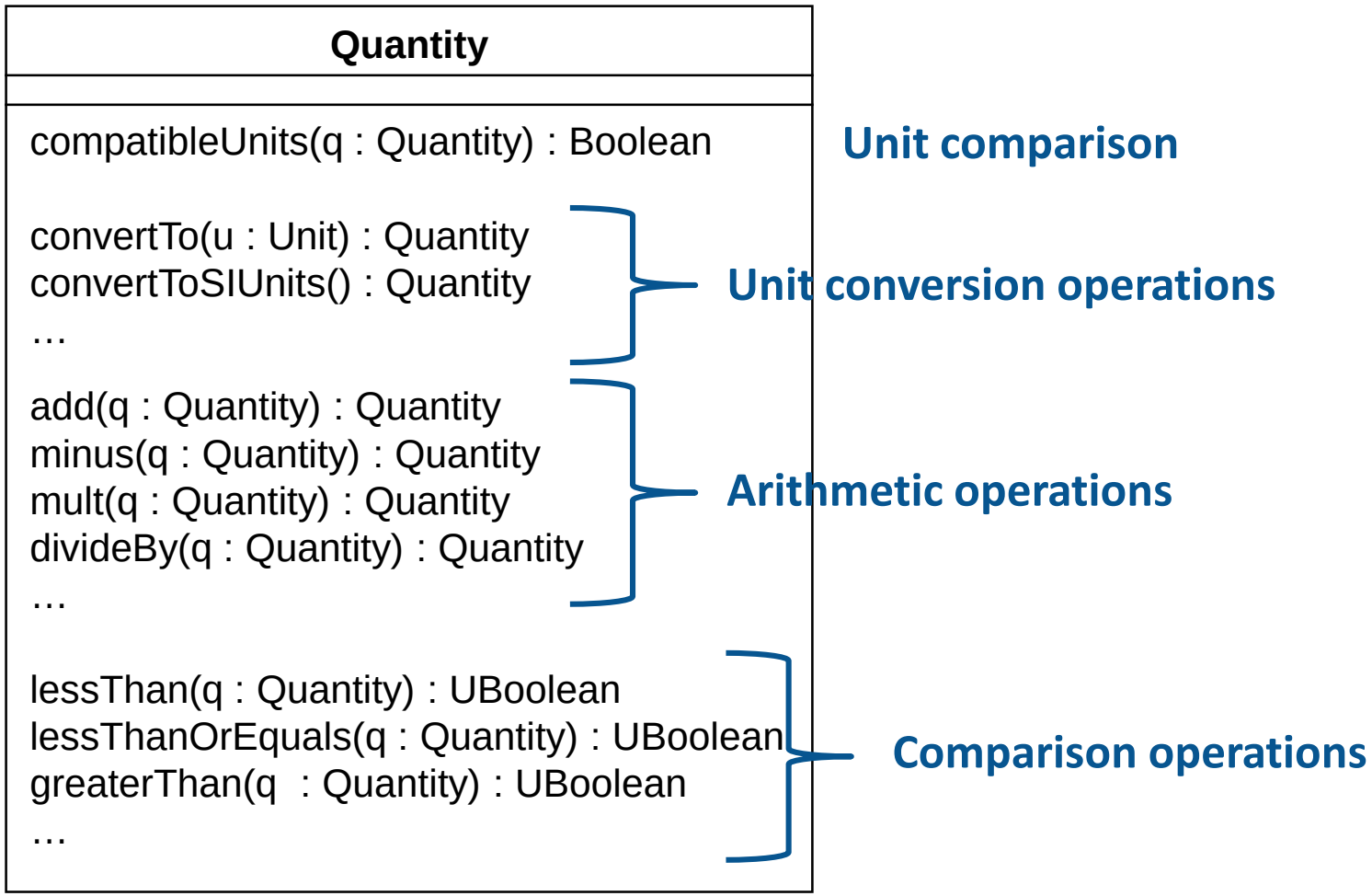

UReal
add(r : UReal) : UReal
minus(r : UReal) : UReal
multiply(r : UReal) : UReal
divideBy(r : UReal) : UReal
power(s : Real) : UReal
…
lessThan(r : UReal) : UBoolean
lessThanOrEquals(r : UReal) : UBoolean
greaterThan(r : UReal) : UBoolean
…
Arithmetic operations
Comparison operations

- Domain Model

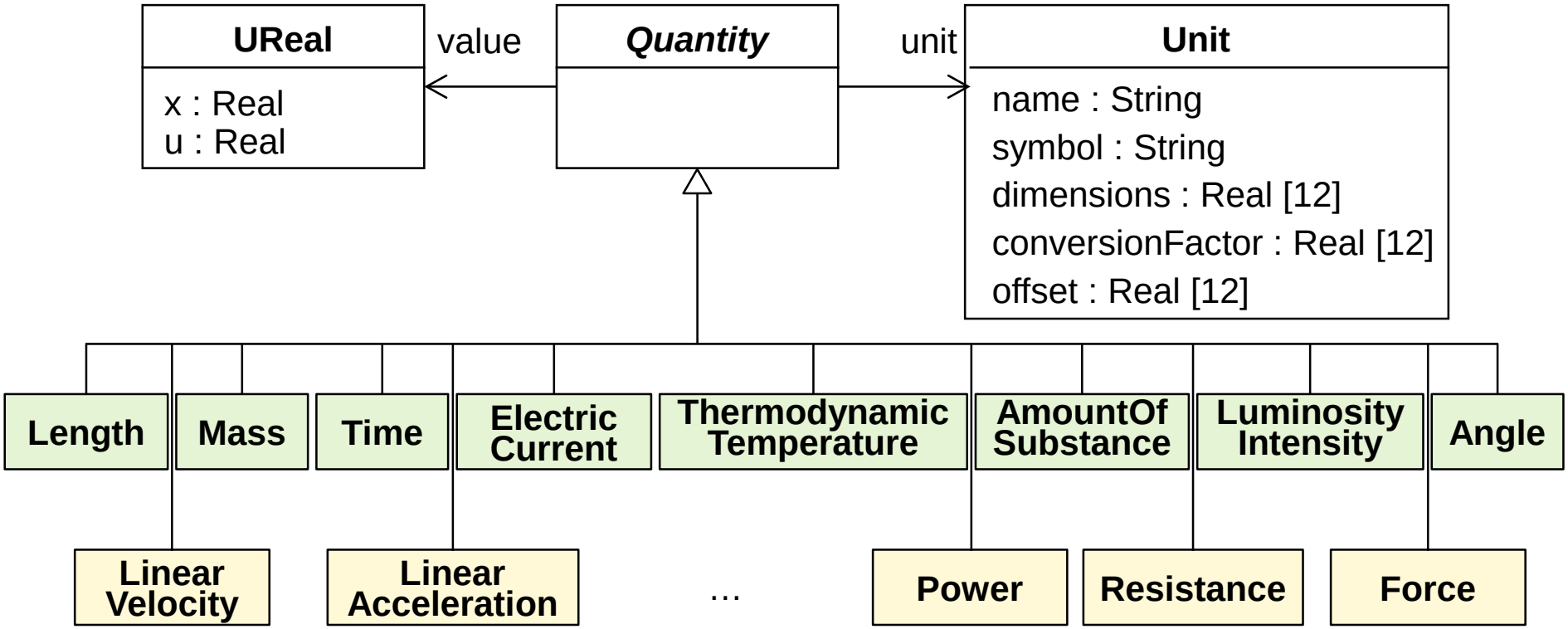


- *Example Instance:* $Length\ 10 \pm 0.001\ m$

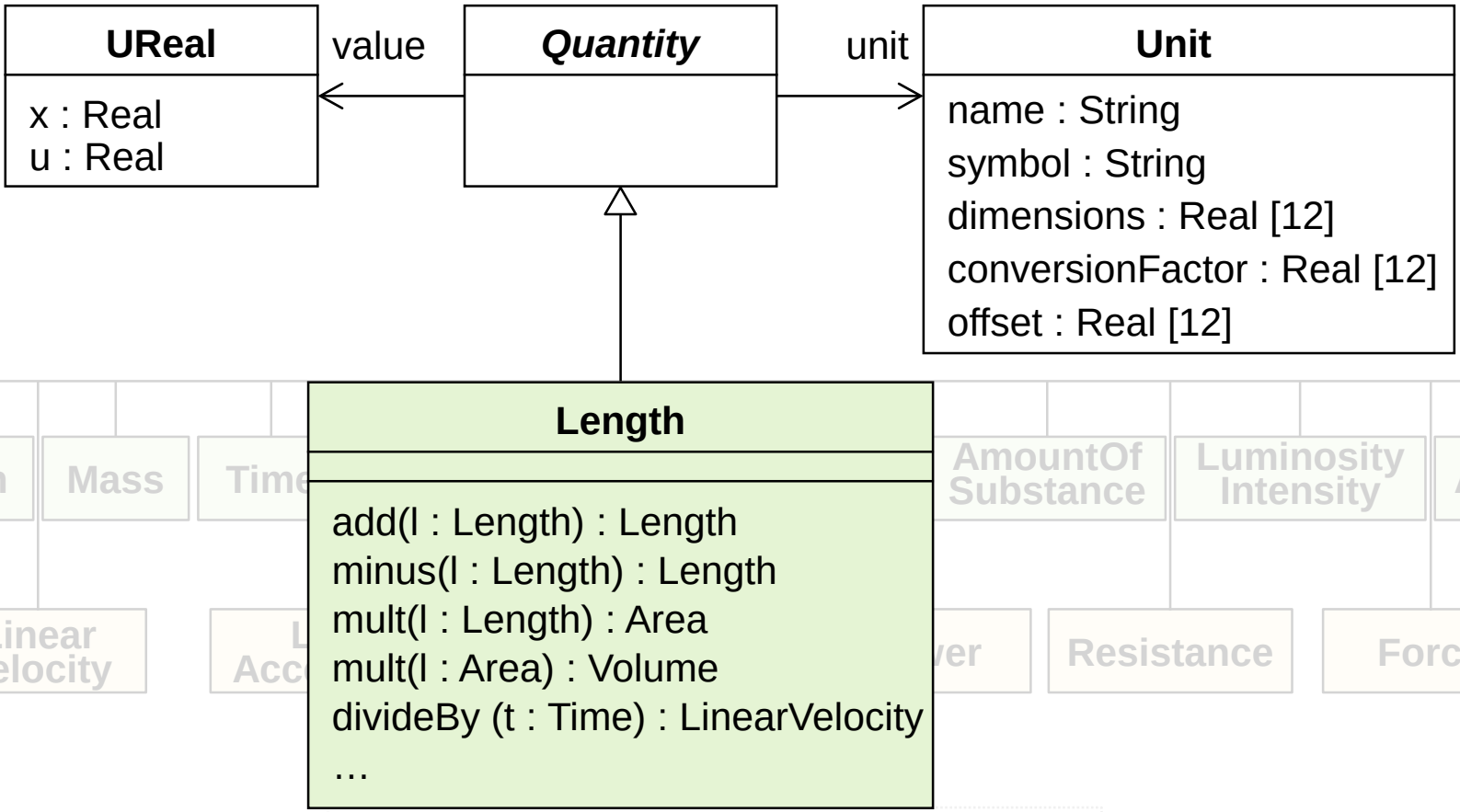



Quantity
compatibleUnits(q : Quantity) : Boolean
convertTo(u : Unit) : Quantity
convertToSIUnits() : Quantity
…
add(q : Quantity) : Quantity
minus(q : Quantity) : Quantity
mult(q : Quantity) : Quantity
divideBy(q : Quantity) : Quantity
…
lessThan(q : Quantity) : UBoolean
lessThanOrEquals(q : Quantity) : UBoolean
greaterThan(q : Quantity) : UBoolean
…
Unit comparison
Unit conversion operations
Arithmetic operations
Comparison operations

- Domain Model



- Domain Model



Implementation

Available Implementations

- Java: Reference implementation
- OCL (USE Tool):
  - Specification of operations with preconditions and postconditions
  - Support for imperative use of operations (SOIL)
- UML (Papyrus, MagicDraw):
  - Support for specifying quantities and computations with quantities
  - Proof-of-concept prototype for executing computations with quantities with fUML
- Download: https://github.com/moliz/moliz.quantitytypes

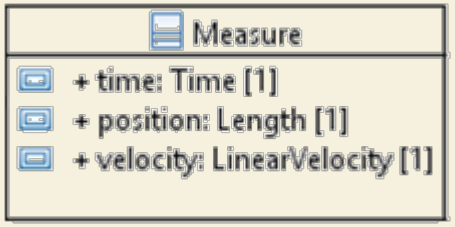
Java Example

```
Length initialPosition = new Length(0, 0.001, Units.Meter);
Length finalPosition = new Length(10, 0.001, Units.Meter);
Length distance = finalPosition.minus(initialPosition);
```

USE OCL Example

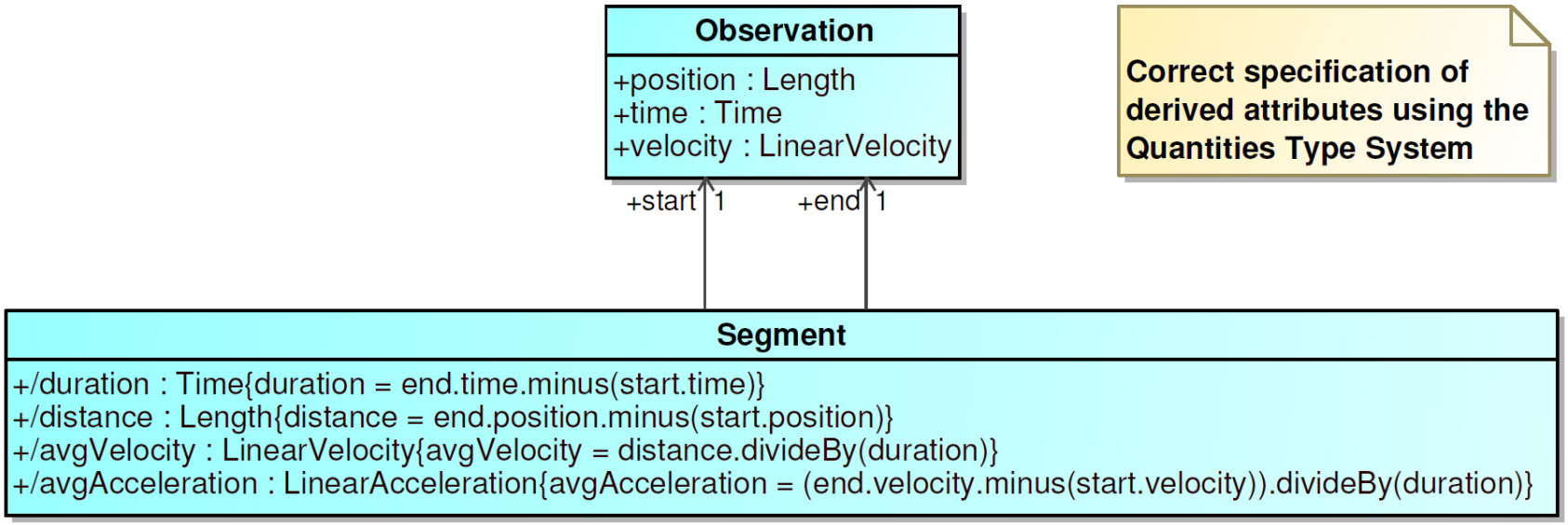
```
!new UReal('ip')
!ip.x : = 0.0
!ip.u := 0.001
!new Quantity('initialPosition')
!initialPosition.value := ip
...
!distance := finalPosition.minus(initialPosition)
```

Papyrus UML Example



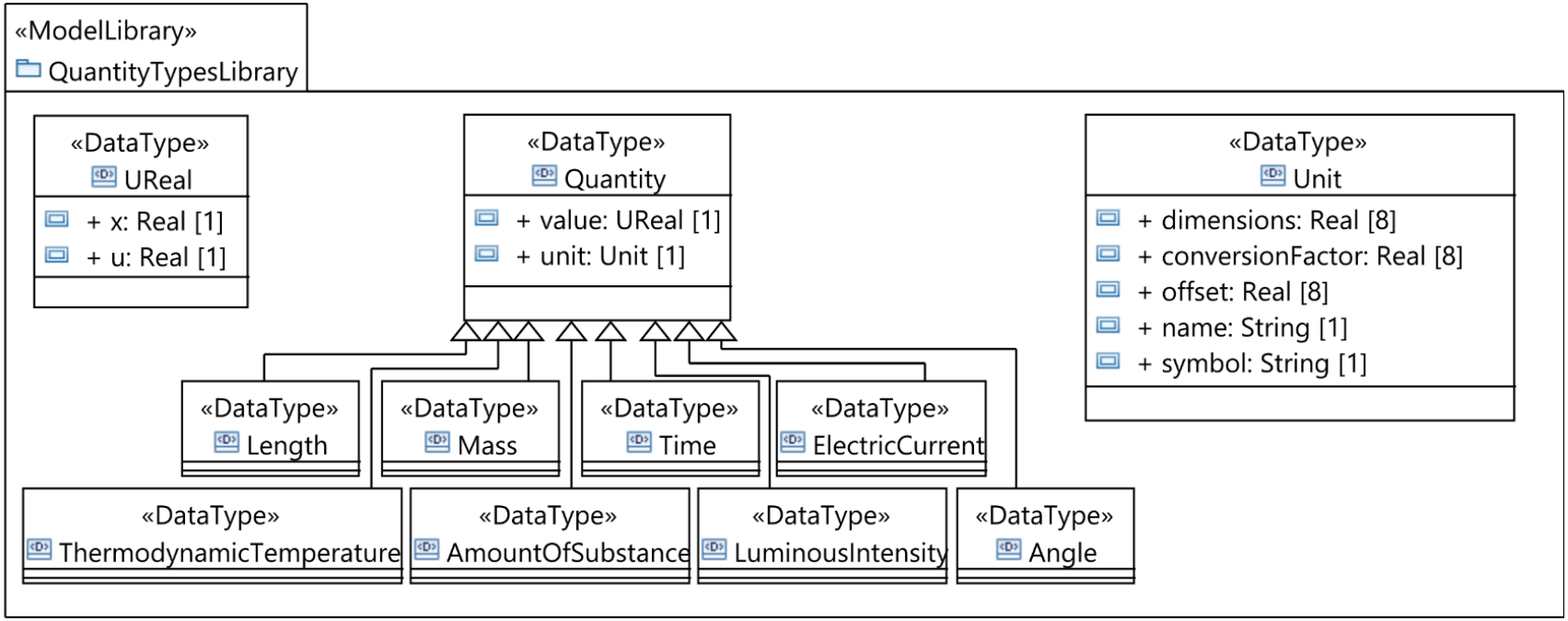
USE Tool: https://sourceforge.net/projects/useocl/ MagicDraw: http://www.nomagic.com/products/magicdraw.html
Eclipse Papyrus UML: https://eclipse.org/papyrus/

UML Implementation



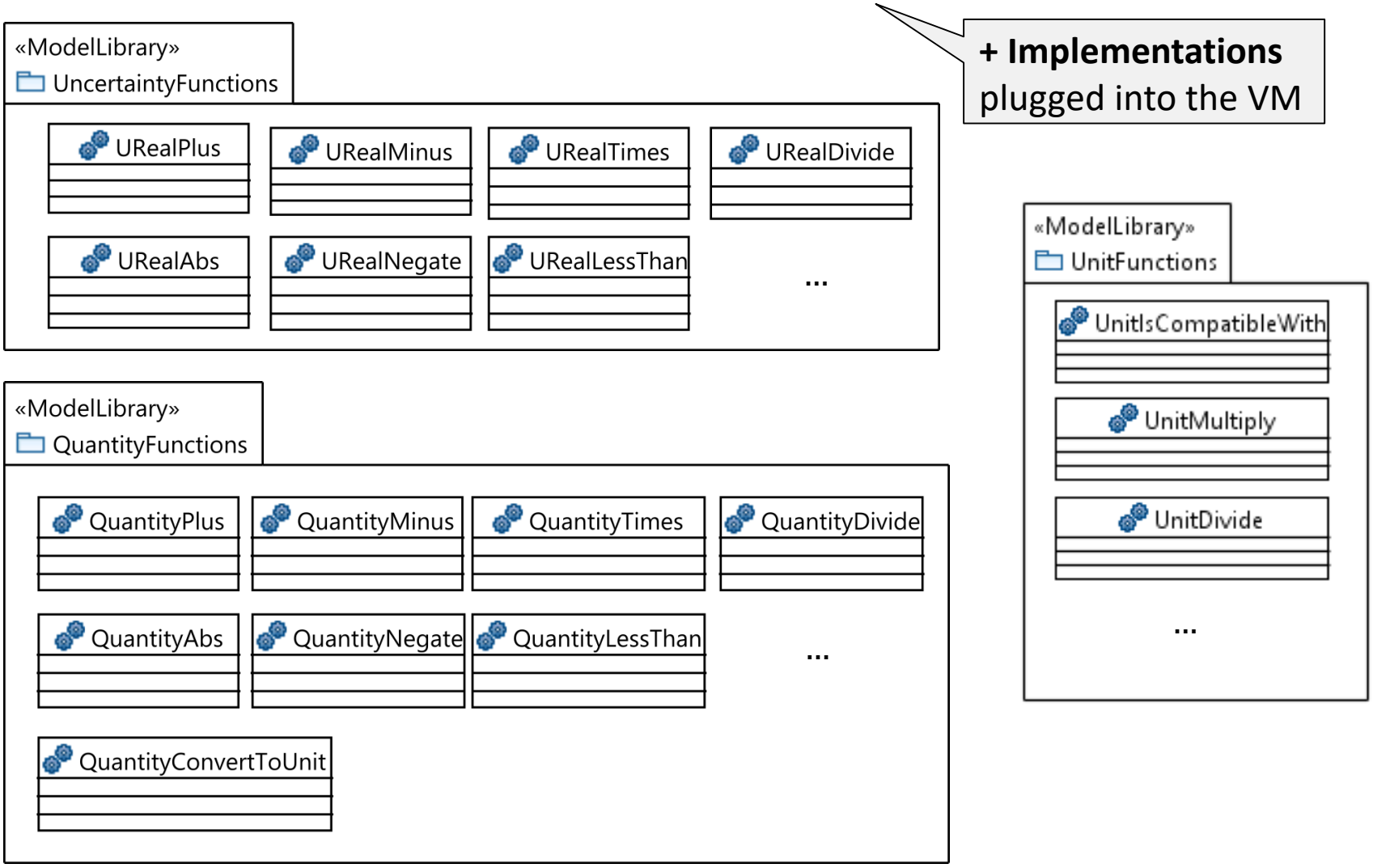
- *Quantities* as types for values (providing both unit and uncertainty info).
- Beyond "decorative" information:
  - Quantities provide methods for aggregating uncertainty values
  - Standard derivation rules for operations on values of these types
  - Static type-checking of operations with quantities and automatic conversion of units (no unit-mismatch problem)

Types


«ModelLibrary»
QuantityTypesLibrary
«DataType»
UReal
+ x: Real [1]
+ u: Real [1]
«DataType»
Quantity
+ value: UReal [1]
+ unit: Unit [1]
«DataType»
Unit
+ dimensions: Real [8]
+ conversionFactor: Real [8]
+ offset: Real [8]
+ name: String [1]
+ symbol: String [1]
«DataType»
Length
«DataType»
Mass
«DataType»
Time
«DataType»
ElectricCurrent
«DataType»
ThermodynamicTemperature
«DataType»
AmountOfSubstance
«DataType»
LuminousIntensity
«DataType»
Angle

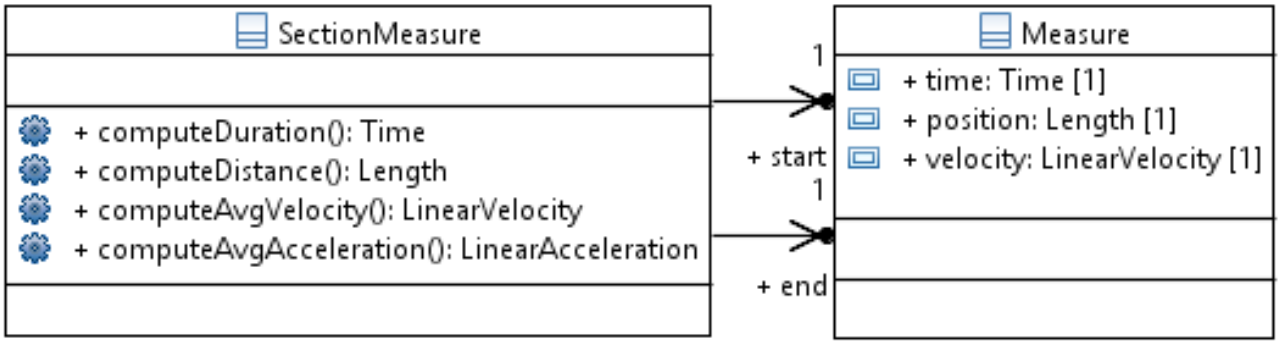
fUML Implementation

Operations: Realized as function behaviors (syntactical elements)

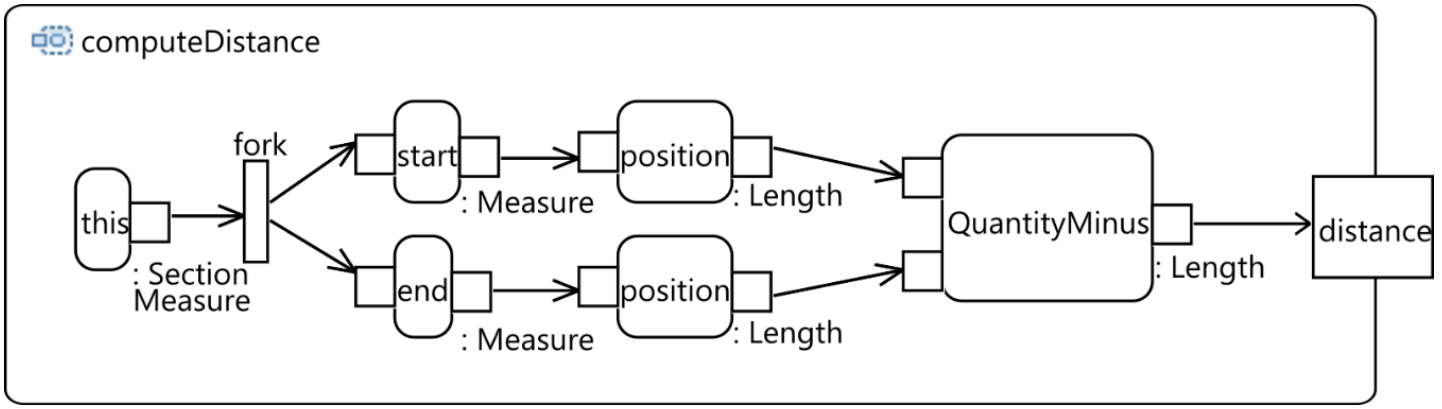


Example

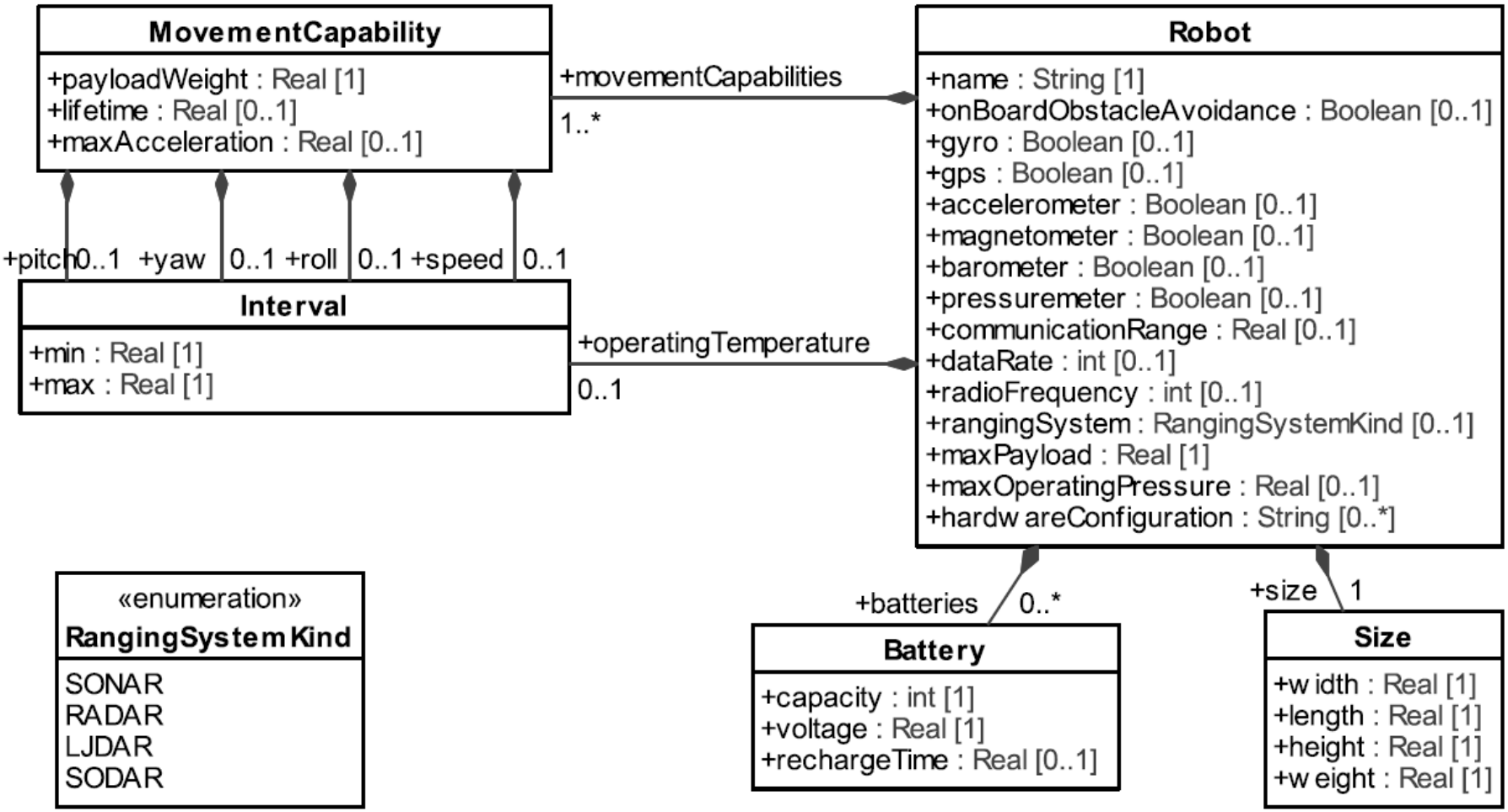
Classes



Behavior

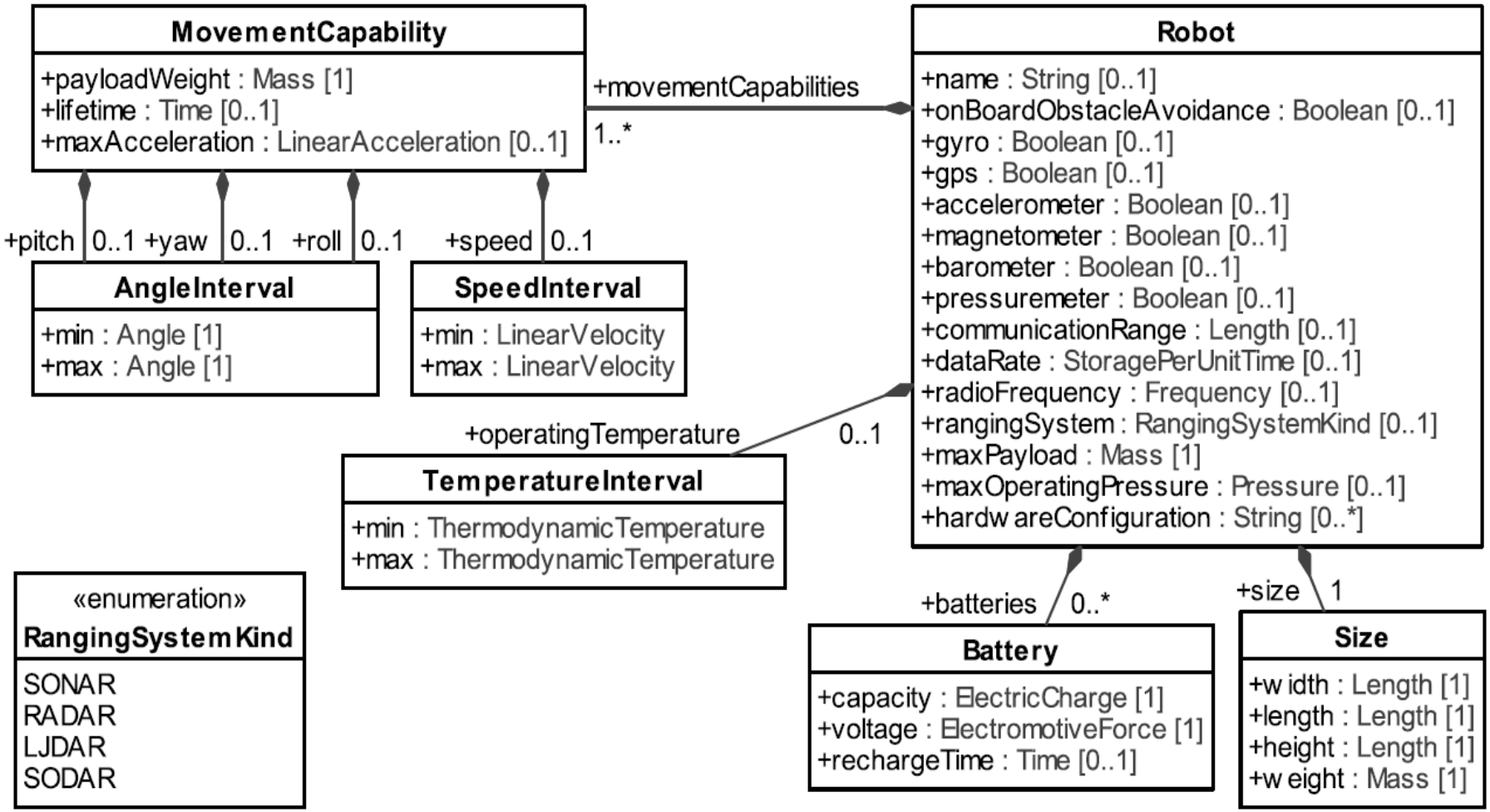


A Family of Robot Languages (Original)

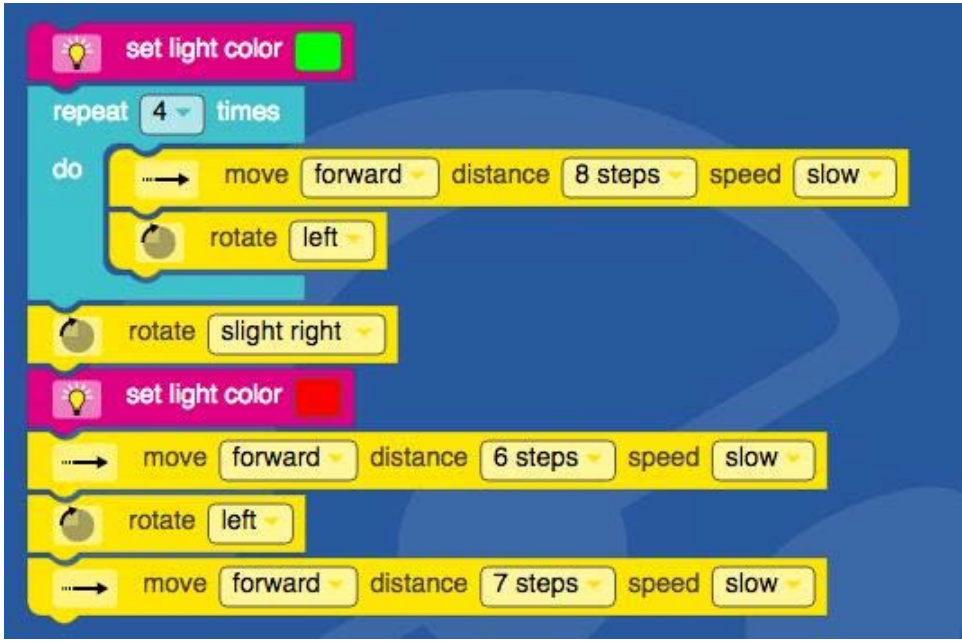


[di Ruscio et al, IEEE Access, 2017]

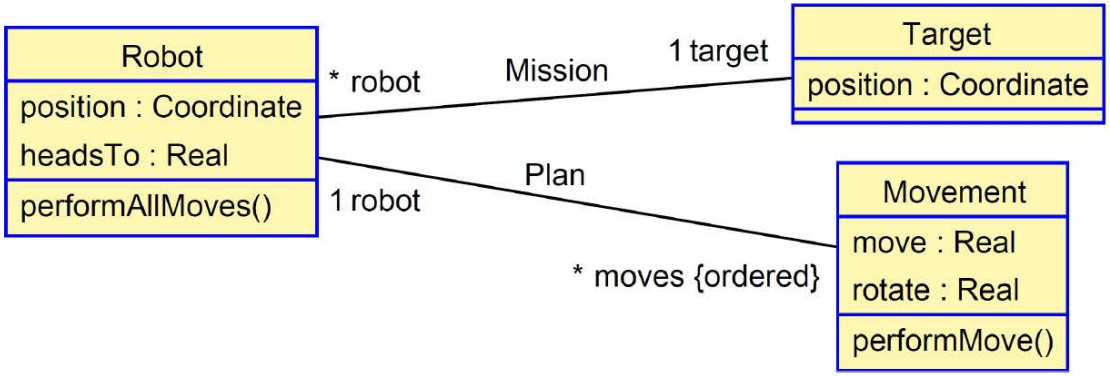
A Family of Robot Languages (with Dimensions)

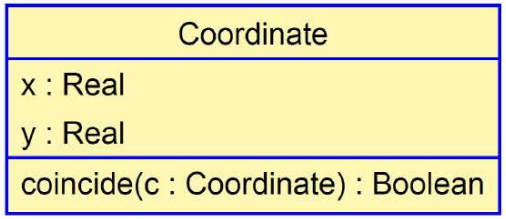


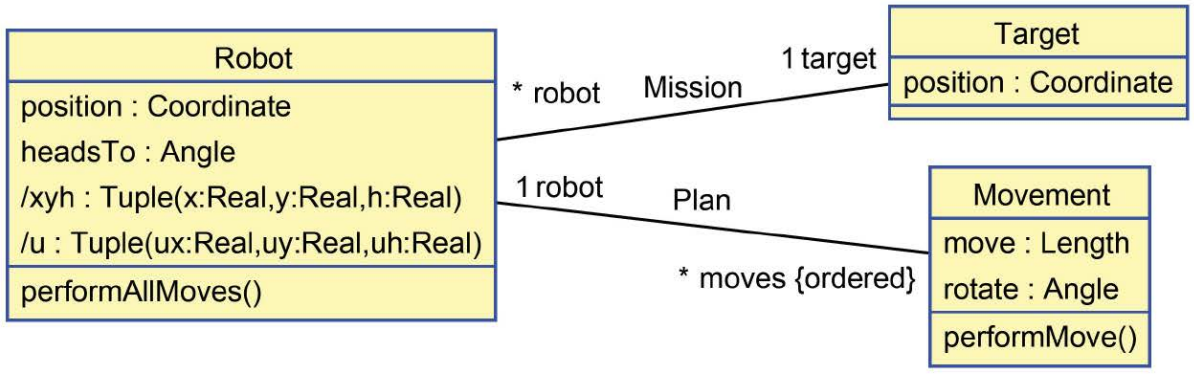
https://ozobot.com/ y https://games.ozoblockly.com/

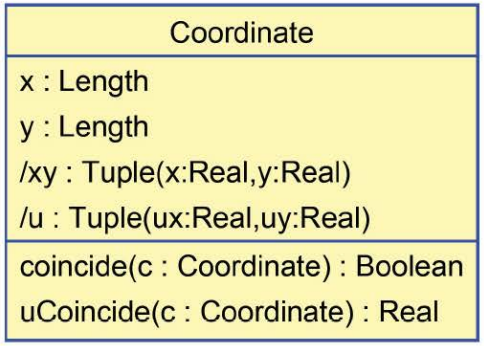


Modeling the robot with USE









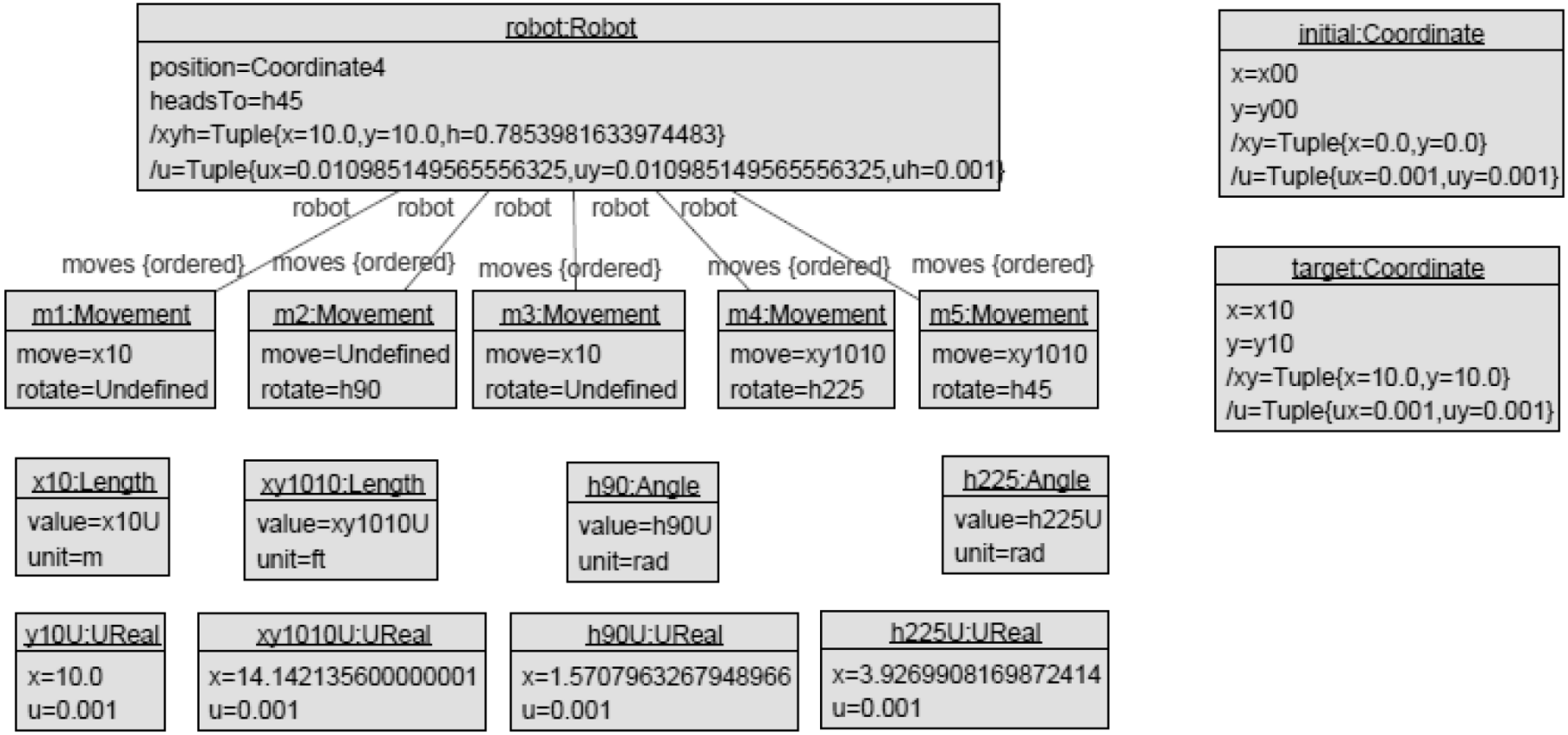
Behaviour in USE

```
class Movement
attributes
 move:Length
 rotate:Angle
operations
 performMove()
 begin
   declare aux:Coordinate,sa:UReal,ca:UReal,dx:Length,dy:Length;
   -- we change the angle first (if we have to)
   if not self.rotate.oclIsUndefined() then
       self.robot.headsTo:=self.rotate
   end;
   -- and then we move (if we have to)
   if not self.move.oclIsUndefined() then
       ca:=self.robot.headsTo.cos();
       sa:=self.robot.headsTo.sin();
       dx:=self.move.sMult(ca);
       dy:=self.move.sMult(sa);
       aux := new Coordinate;
       aux.x := self.robot.position.x.add(dx);
       aux.y := self.robot.position.y.add(dy);
       self.robot.position:=aux;
    end;
  end
end
```

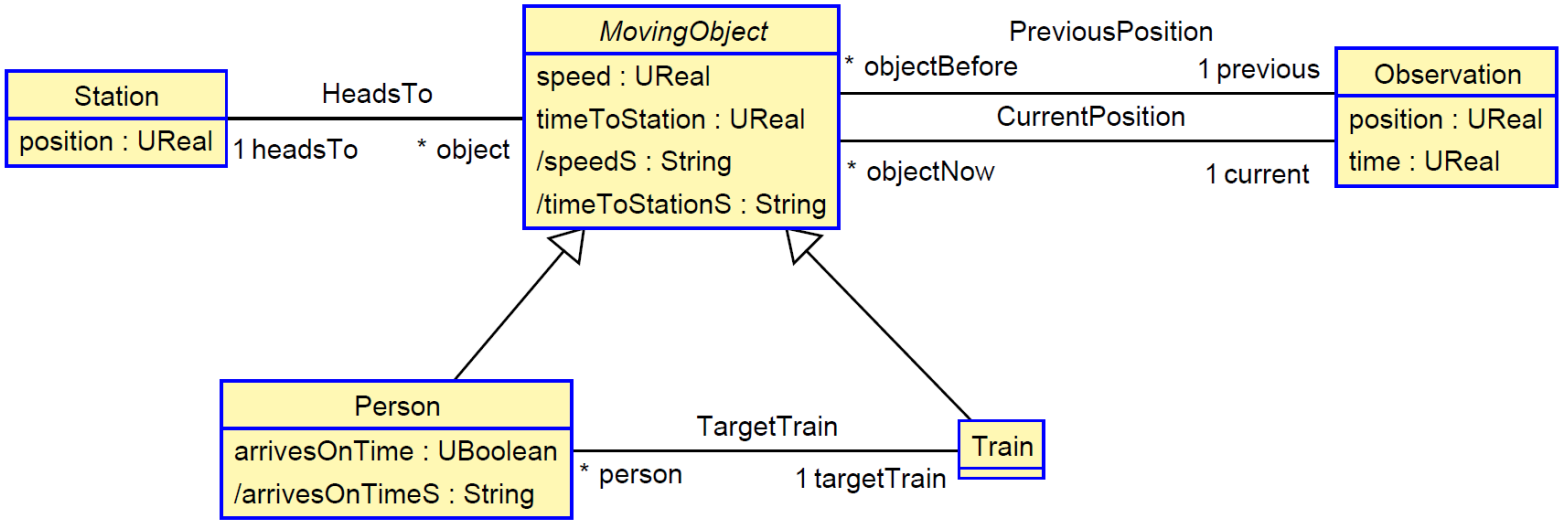
Executing the USE specs

```
Robot.soil> !new Coordinate('initial')
Robot.soil> !initial.x:=x00
Robot.soil> !initial.y:=y00
Robot.soil> !new Coordinate('target')
Robot.soil> !target.x:=x10
Robot.soil> !target.y:=y10
Robot.soil> !new Robot('robot')
Robot.soil> !robot.position:=initial
Robot.soil> !robot.headsTo:=h0
Robot.soil> !new Movement('m1')
Robot.soil> !m1.move:=x10
Robot.soil> !new Movement('m2')
Robot.soil> !m2.rotate:=h90
Robot.soil> !new Movement('m3')
Robot.soil> !m3.move:=x10
Robot.soil> !new Movement('m4')
Robot.soil> !m4.rotate:=h225
Robot.soil> !m4.move:=xy1010
Robot.soil> !new Movement('m5')
Robot.soil> !m5.rotate:=h45
Robot.soil> !m5.move:=xy1010
Robot.soil> !insert(robot,m1) into Plan
Robot.soil> !insert(robot,m2) into Plan
Robot.soil> !insert(robot,m3) into Plan
Robot.soil> !insert(robot,m4) into Plan
Robot.soil> !insert(robot,m5) into Plan
Robot.soil> !robot.performAllMoves()
Robot.soil> !r:=robot.position.uCoincide(target)
Robot.soil> ?r
-> 0.03479626661931806 : Real
```

Object diagram with the resulting system



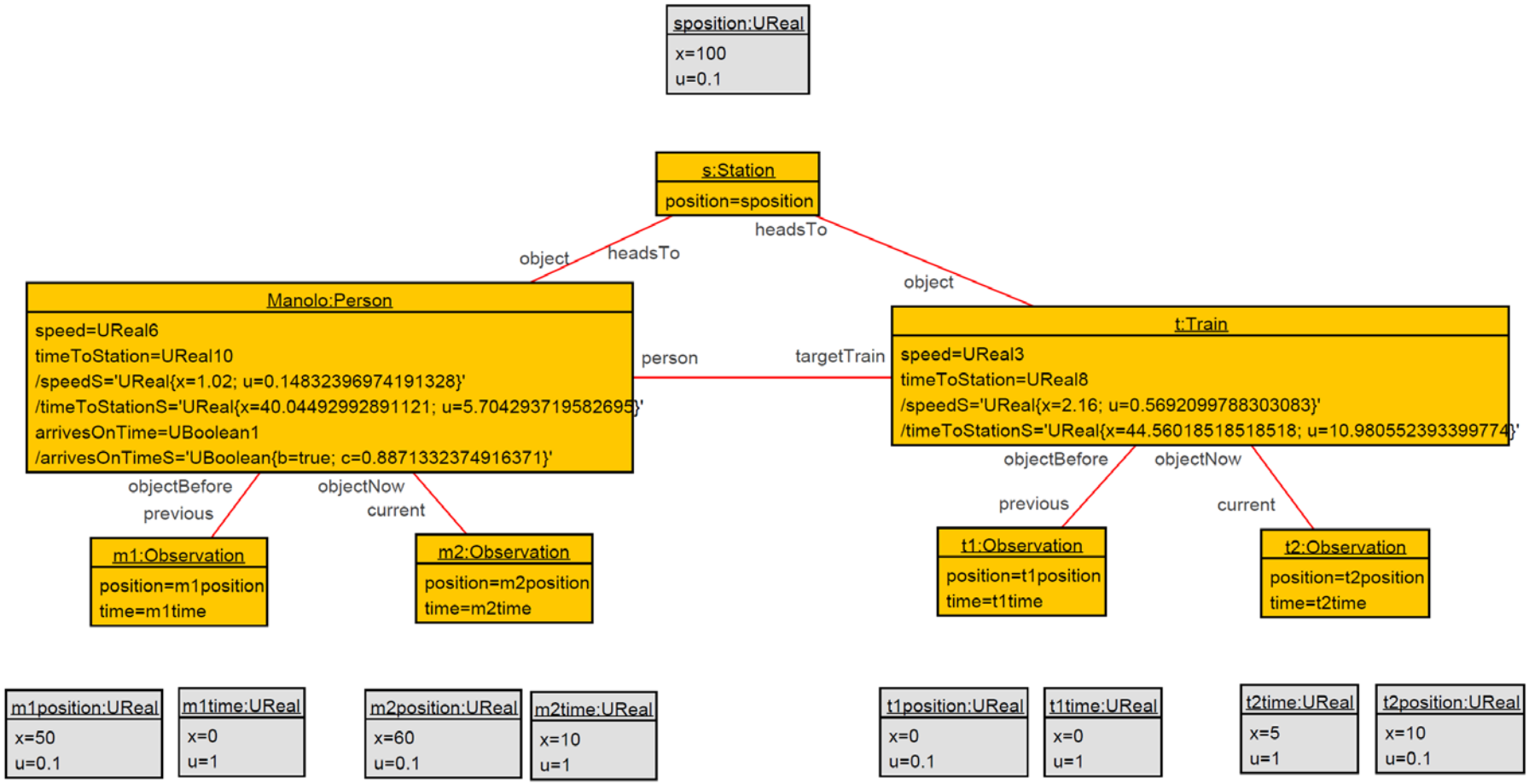
Integración con el sistema de tipos de OCL (USE)




```
abstract class MovingObject
attributes
  speed:UReal derive: (self.current.position-self.previous.position)/(self.current.time-self.previous.time)
  timeToStation: UReal derive: (self.headsTo.position-self.current.position)/self.speed
  speedS :String derive: self.speed.toString()
  timeToStationS :String derive: self.timeToStation.toString()
end

class Person < MovingObject
attributes
  arrivesOnTime:UBoolean derive: (self.timeToStation + 3) < self.targetTrain.timeToStation
  arrivesOnTimeS:String derive: self.arrivesOnTime.toString()
end
```

Integración con el sistema de tipos de OCL (USE)



Ongoing and Future Work

- **Implementation**
  - Evolve fUML proof-of-concept implementation to full implementation
  - Alf implementation (textual action language for fUML)
  - Full integration with Papyrus (already done for MagicDraw)
  - Integration with the OCL/USE type system
- **Refinement of the conceptual model of quantity types**
  - Different kinds of uncertainty (e.g., interval, different probability distributions)
- **Representation of quantities**
  - More usable representation of quantities
  - Integration with existing standards, e.g., MARTE and SysML
  - Connection with other analysis/simulation tools (Modelica, Simulink, etc.)
- **More extensive case studies**
  - Larger case studies to evaluate the usability and effectiveness of our notation
  - Industrial case studies to analyze the propagation of uncertainty in real situations

Thank You!

Questions?

Contact

Tanja Mayerhofer
mayerhofer@big.tuwien.ac.at

Manuel Wimmer
wimmer@big.tuwien.ac.at

Antonio Vallecillo
av@lcc.uma.es

Loli Burgueño
loli@lcc.uma.es

References

A. Vallecillo, C. Morcillo, and P. Orue. Expressing Measurement Uncertainty in Software Models. In Proc. of 10th Int. Conf. on the Quality of Information and Communications Technology (QUATIC), 1–10, 2016.

T. Mayerhofer, M. Wimmer, A. Vallecillo. Adding Uncertainty and Units to Quantity Types in Software Models. In Proc. of 2016 ACM SIGPLAN Int. Conf. on Software Language Engineering (SLE), ACM, 118–131, 2016.

Code Repository: https://github.com/moliz/moliz.quantitytypes

Web Page: http://atenea.lcc.uma.es/index.php/Main_Page/Resources/DataUncertainty

BIG TU WIEN
Business Informatics Group

Atenea
Modeling Software Systems

UNIVERSIDAD DE MÁLAGA