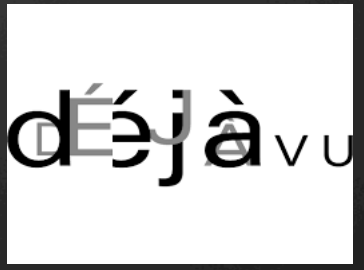
K

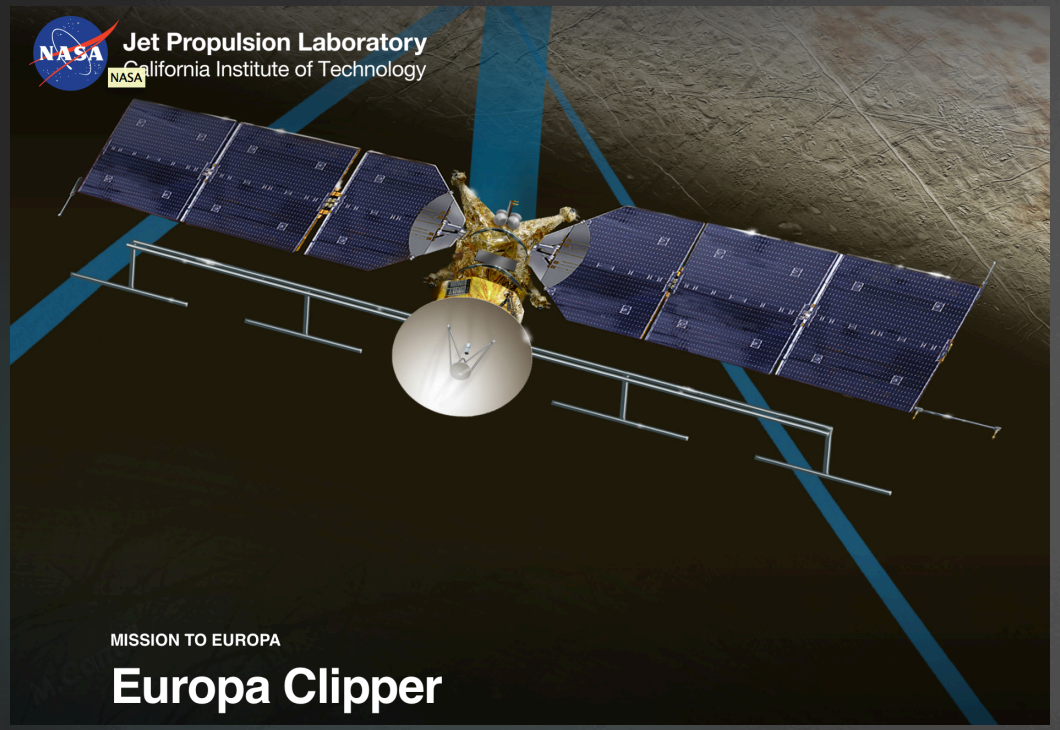
A WIDE SPECTRUM TEXTUAL LANGUAGE FOR MODELING AND IMPLEMENTATION

Bradley Clement, Chris Delp, Klaus Havelund, Rahul Kumar

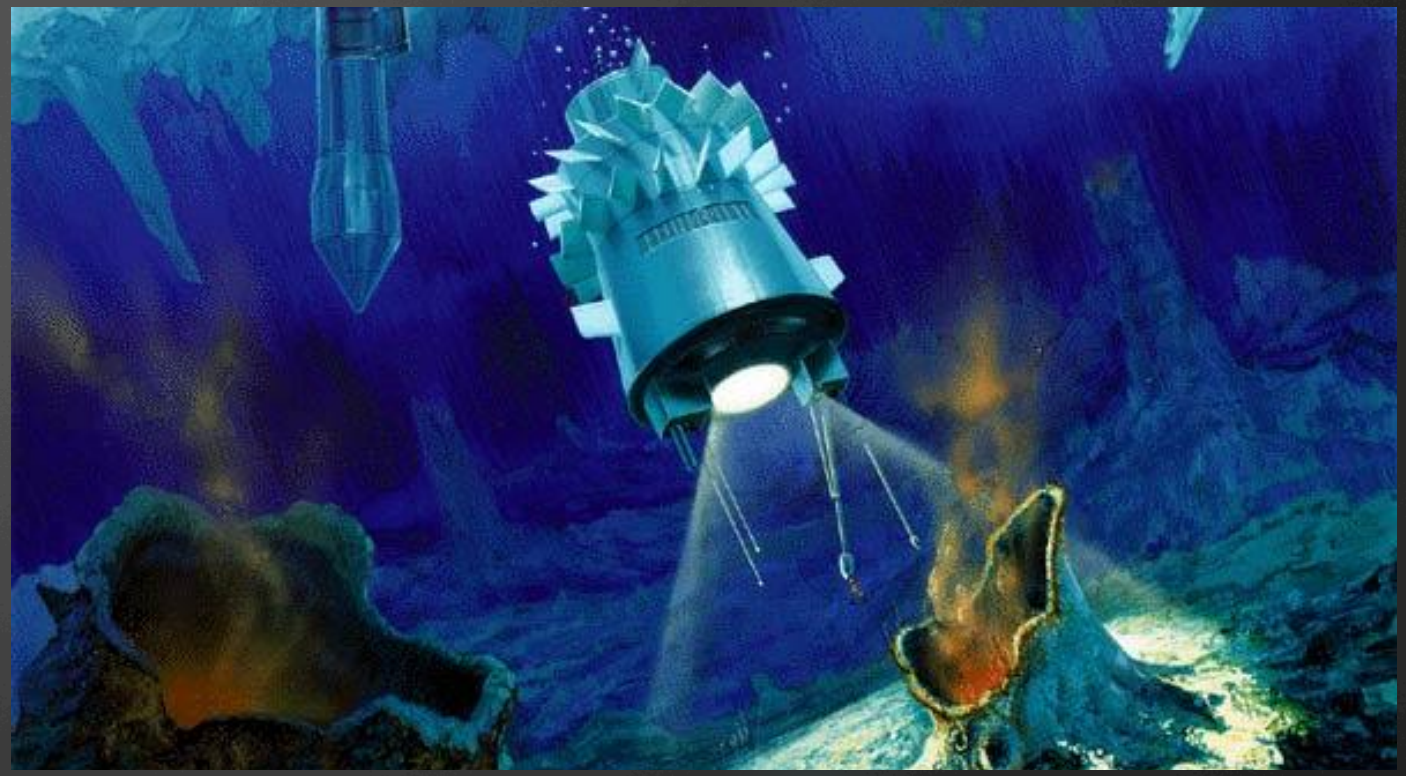


Going to Europa

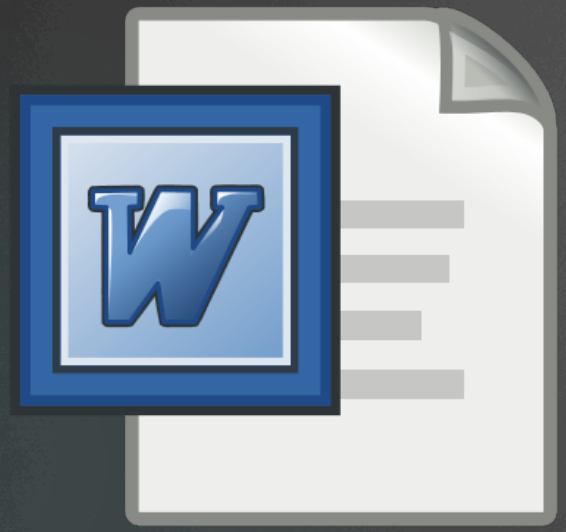
The Europa Clipper mission



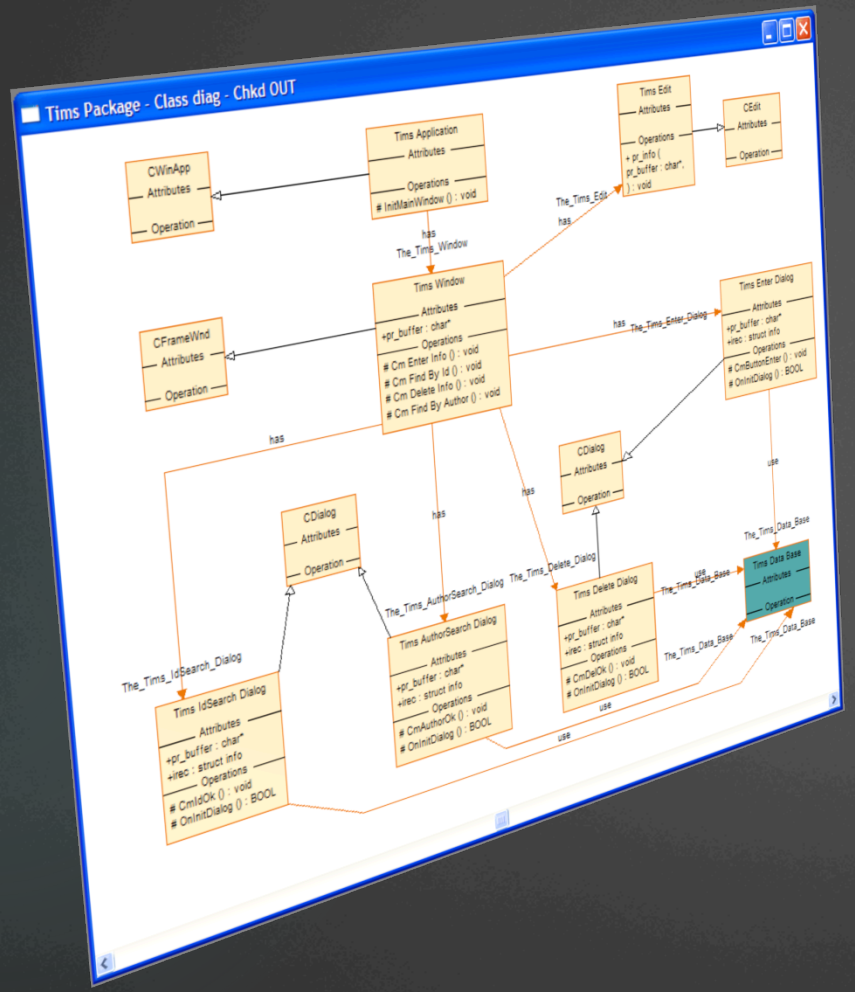
Perhaps in some future:



How are missions normally designed?

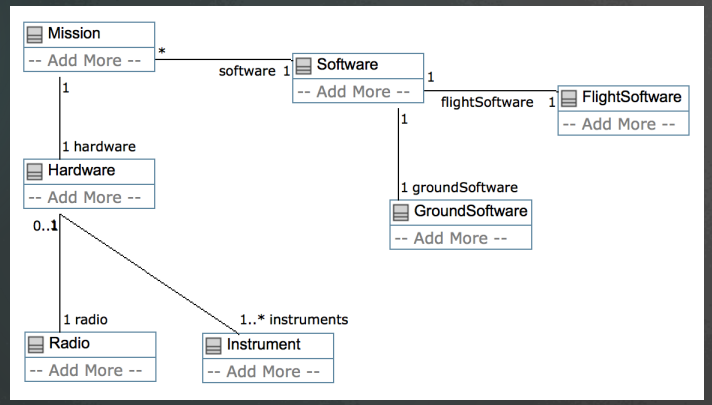


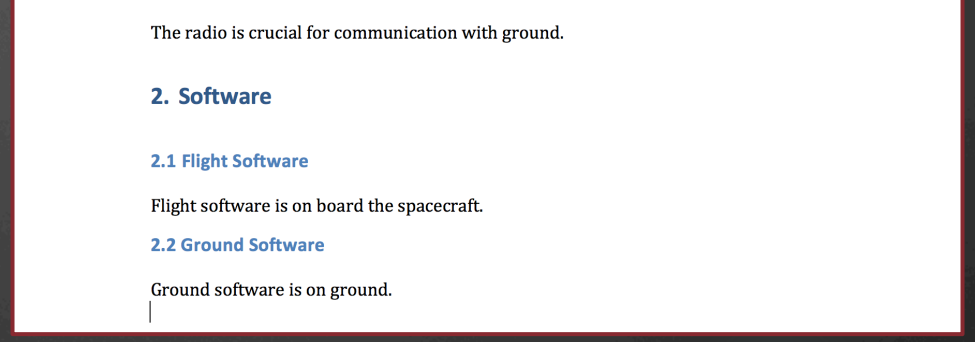
Alterative pursued @ JPL



UML/SysML

A path to acceptance





Mission

1. Hardware

1.1 Instruments

Instruments provide science results.

1.2 Radio

The radio is crucial for communication with ground.

2. Software

2.1 Flight Software

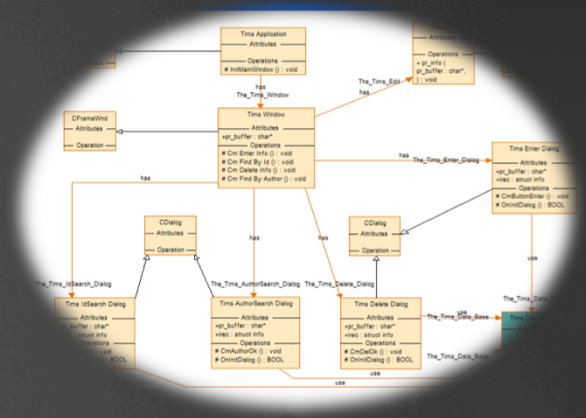
Flight software is on board the spacecraft.

2.2 Ground Software

Ground software is on ground.

Observation 1

popular:
class diagrams + constraints



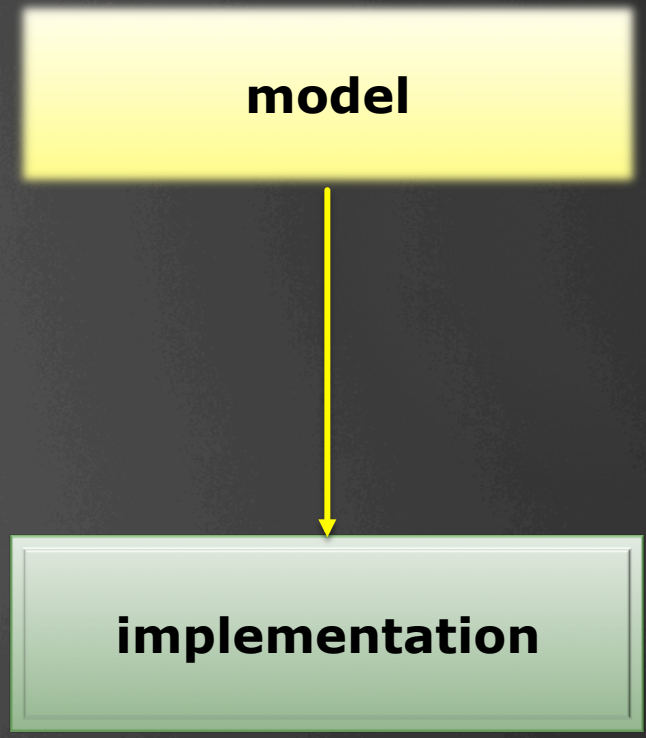
- Model based engineering community @ JPL
  - UML, SysML, Visual languages
  - Semantics difficult to find/use
  - No analysis

- Formal methods community @ JPL
  - Formal and Textual languages
  - Semantics clearly defined
  - Analysis (model checking, th. proving, …)

$$
\begin{array}{l}
project : OZSpec \rightarrow UMLDiagram \\
\hline
\forall (oz, uml) : project \bullet \\
\quad \{c : oz \cap Classdef \bullet c.name\} = \{c : uml.classes \\
\qquad \bullet c.name\} \bullet \forall c_1, c_2 : oz \cap Classdef \bullet \exists_1 c' : \\
\qquad\quad uml.classes \bullet c'.name = c_1.name \\
\qquad\quad c'.attris = \{cls : Classdef \mid cls \in oz \bullet cls.name\} \\
\qquad\qquad \lhd c_1.state.decpart \\
\qquad\quad c'.ops = \{o : Opdef \mid o \in c_1.ops \bullet o.name\} \\
\qquad c_2.name \in \{t : \mathrm{ran}\ c_1.state.decpart \bullet t.name\} \Rightarrow \\
\qquad\quad \exists_1 (c'_1, c'_2) : uml.agg \bullet c'_1.name = c_1.name \\
\qquad\qquad \wedge\ c'_2.name = c_2.name \\
\qquad c_2.name \in \{inh : \mathrm{dom}\ c_1.inherit \bullet inh.name\} \Rightarrow \\
\qquad\quad \exists_1 (c'_1, c'_2) : uml.inh \bullet c'_1.name = c_1.name \\
\qquad\qquad \wedge\ c'_2.name = c_2.name
\end{array}
$$

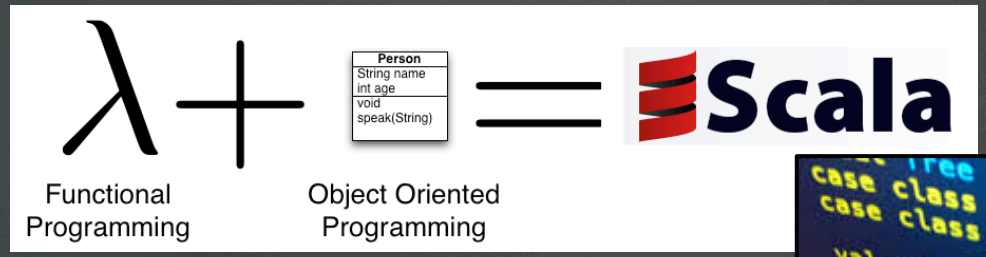
But the goals are the same

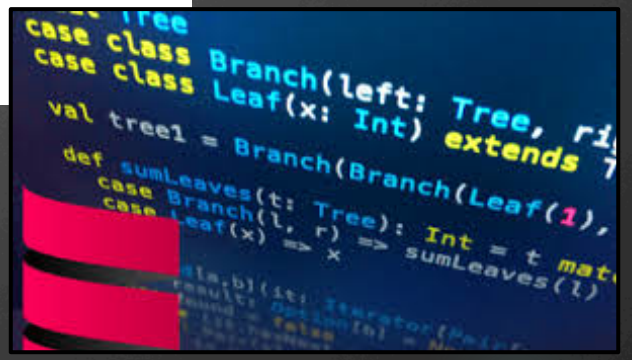
- Create models
- Analyze for correctness
- Possibly produce implementation
- Verify implementation



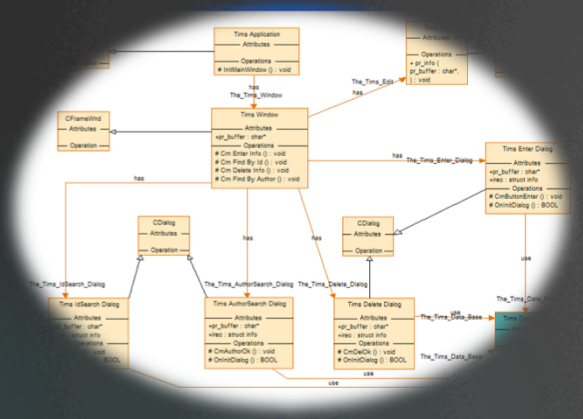
Observation 2

- Programming languages
  - Modern programming languages moving in the direction of spec L.
  - Combining object oriented and functional programming
  - Supporting collections, such as sets, lists and maps





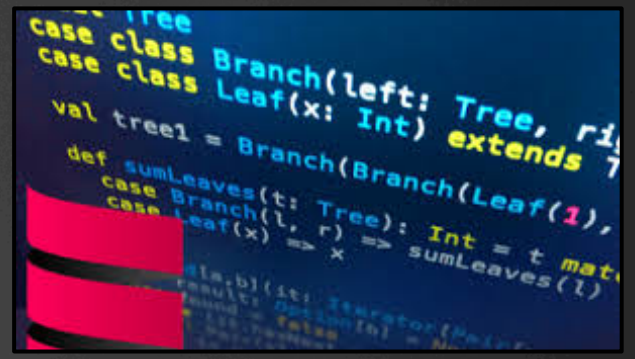
Conclusion



+

$$
\begin{array}{l}
project : OZSpec \rightarrow UMLDiagram \\
\hline
\forall (oz, uml) : project \bullet \\
\quad \{c : oz \cap Classdef \bullet c.name\} = \{c : uml.classes \\
\qquad \bullet c.name\} \bullet \forall c_1, c_2 : oz \cap Classdef \bullet \exists_1 c' : \\
\qquad\quad uml.classes \bullet c'.name = c_1.name \\
\qquad\quad c'.attris = \{cls : Classdef \mid cls \in oz \bullet cls.name\} \\
\qquad\qquad \lhd c_1.state.decpart \\
\qquad\quad c'.ops = \{o : Opdef \mid o \in c_1.ops \bullet o.name\} \\
\qquad c_2.name \in \{t : \mathrm{ran}\ c_1.state.decpart \bullet t.name\} \Rightarrow \\
\qquad\quad \exists_1 (c_1', c_2') : uml.agg \bullet c_1'.name = c_1.name \\
\qquad\qquad \wedge\ c_2'.name = c_2.name \\
\qquad c_2.name \in \{inh : \mathrm{dom}\ c_1.inherit \bullet inh.name\} \Rightarrow \\
\qquad\quad \exists_1 (c_1', c_2') : uml.inh \bullet c_1'.name = c_1.name \\
\qquad\qquad \wedge\ c_2'.name = c_2.name
\end{array}
$$

+



Obvious questions

- Why not pick an existing formal specification language?
- Why not pick an existing programming language?

Some possible answers

- Desirable to own the tool stack, full control over syntax, parser, type checker, analysis support.
- Keeping it as simple as possible. Scala is complex, for tool builders and for modelers.
- Full automation required. No interactive theorem proving.

- But: there is no really good reason for not choosing a programming language like for example Scala.

Modeling as programming

- They want to be able to mix modeling and programming, for example by going from requirements to test scripts.

- As management pointed out at a presentation:

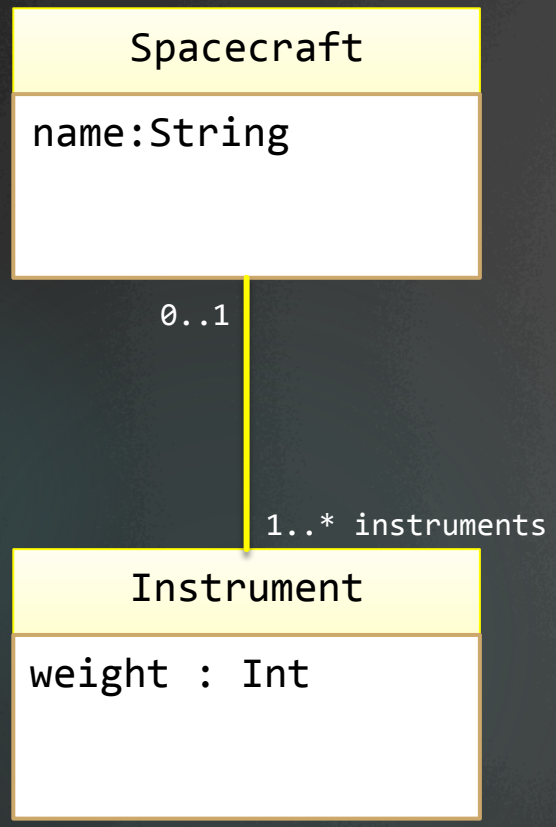
> *Most people are been taught programming at some point and will find it much easier than all this diagramming.*

K Language Constructs

- Packages
- Classes
  - Inheritance
  - Properties (fields): Bool, Int, Real, Sets, Bags, Lists, …
  - Functions (with subtyping)
  - Constraints
    - Rich expression language (predicate logic)
    - Side effects eventually (programming)
- Relations
  - Multiplicities
- Annotations

On wish list:
Full reflection: write a static analyzer for K in K as a library.

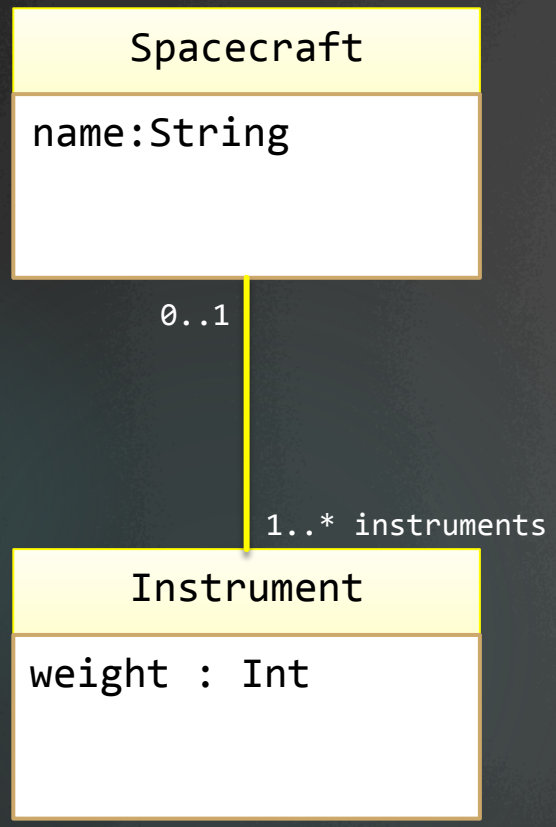
Properties



- Class **Spacecraft** is said to have the **property** “instruments”.
- MBE refers to this as a “relation”
- FM refers to this as a “field”

$$s.instruments = \{i : Instrument \mid (s,i) \in R$$

Relational View



Given are sets of atomic values:

$$SpaceCraft$$
$$Instrument$$

and the relations:

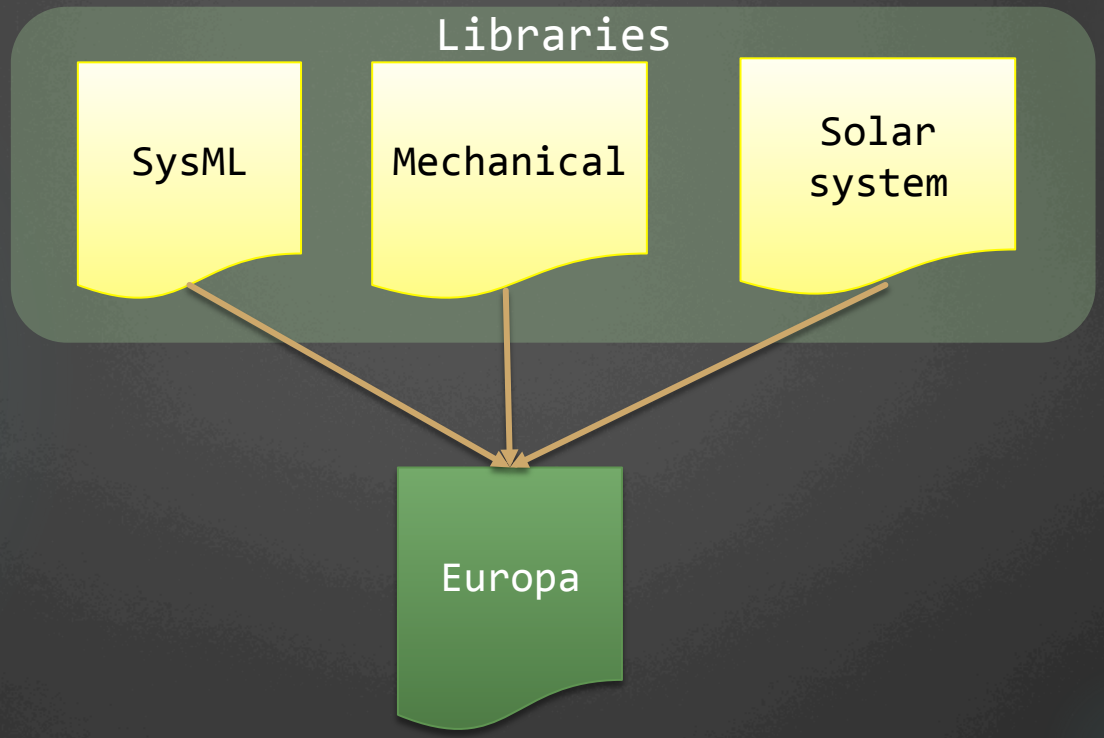
$$R \subseteq Spacecraft \times Instrument$$
$$name \subseteq Spacecraft \times String$$
$$weight \subseteq Instrument \times int$$

with the constraints:

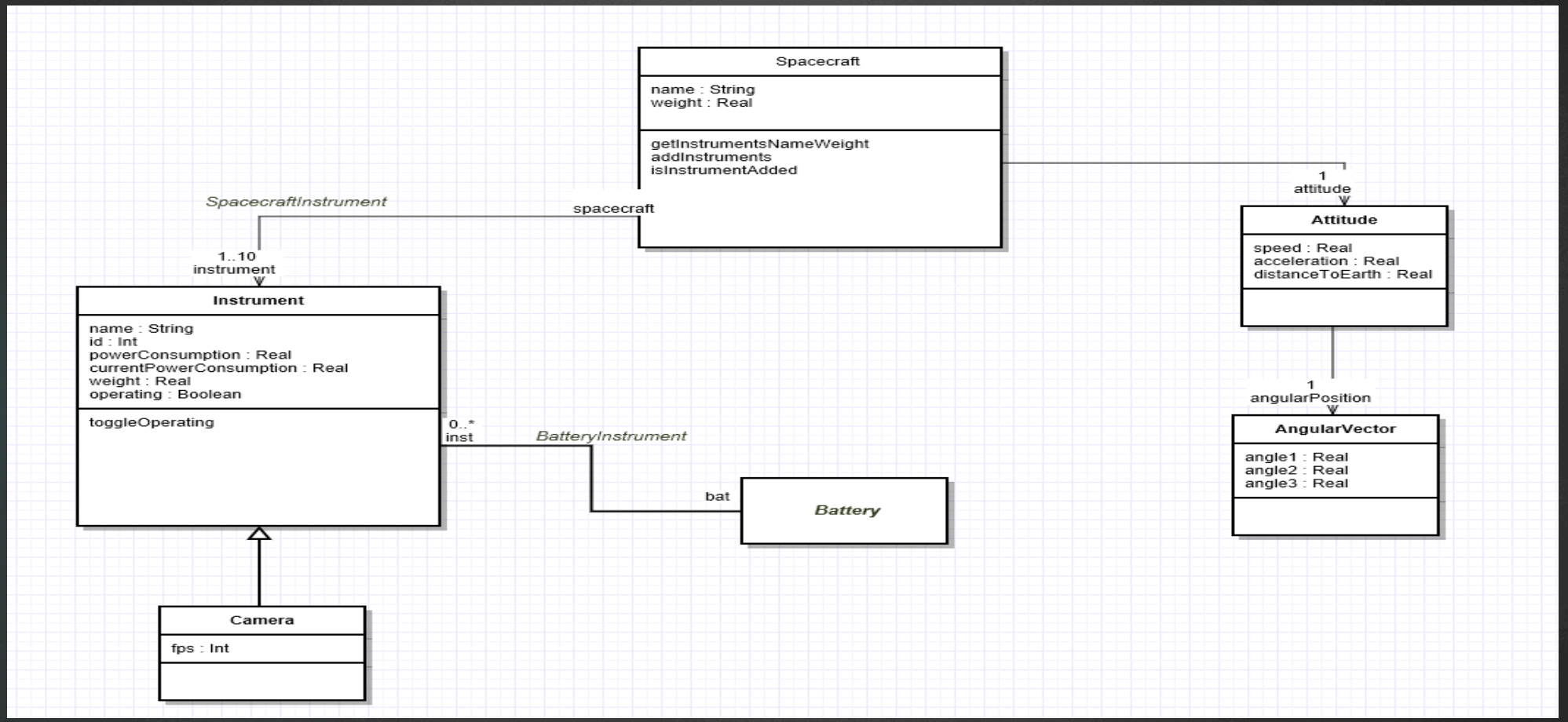
$$\forall i : Instrument \bullet \mathbf{card}\{s : SpaceCraft \mid (s, i) \in R\} \leq 1$$
$$\forall i : Instrument \bullet \mathbf{card}\{x : Int \mid (s, x) \in weight\} = 1$$
$$\forall s : Spacecraft \bullet \mathbf{card}\{x : String \mid (s, x) \in name\} = 1$$

Domains

- Domain specific theories encoded as K libraries

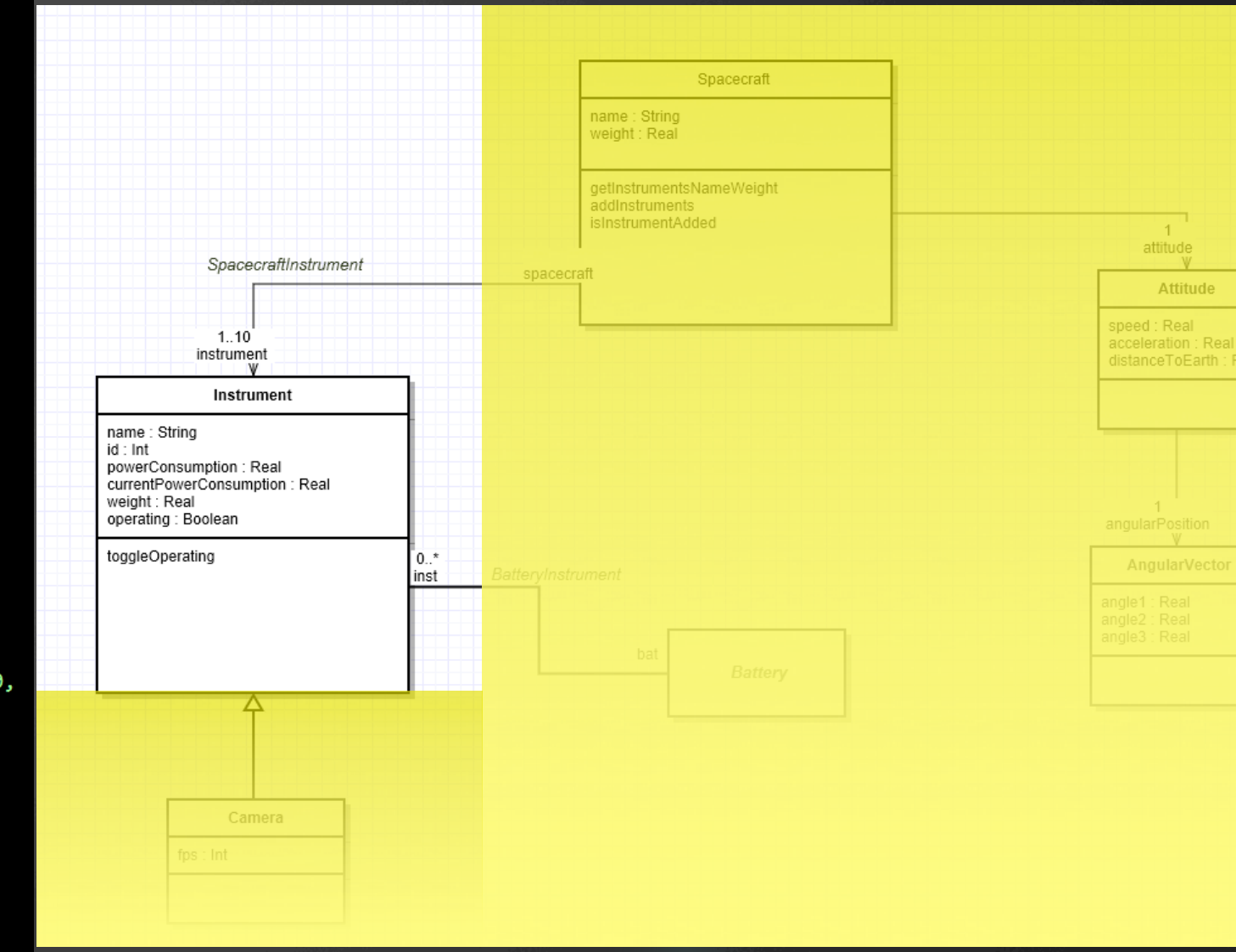


Spacecraft Example



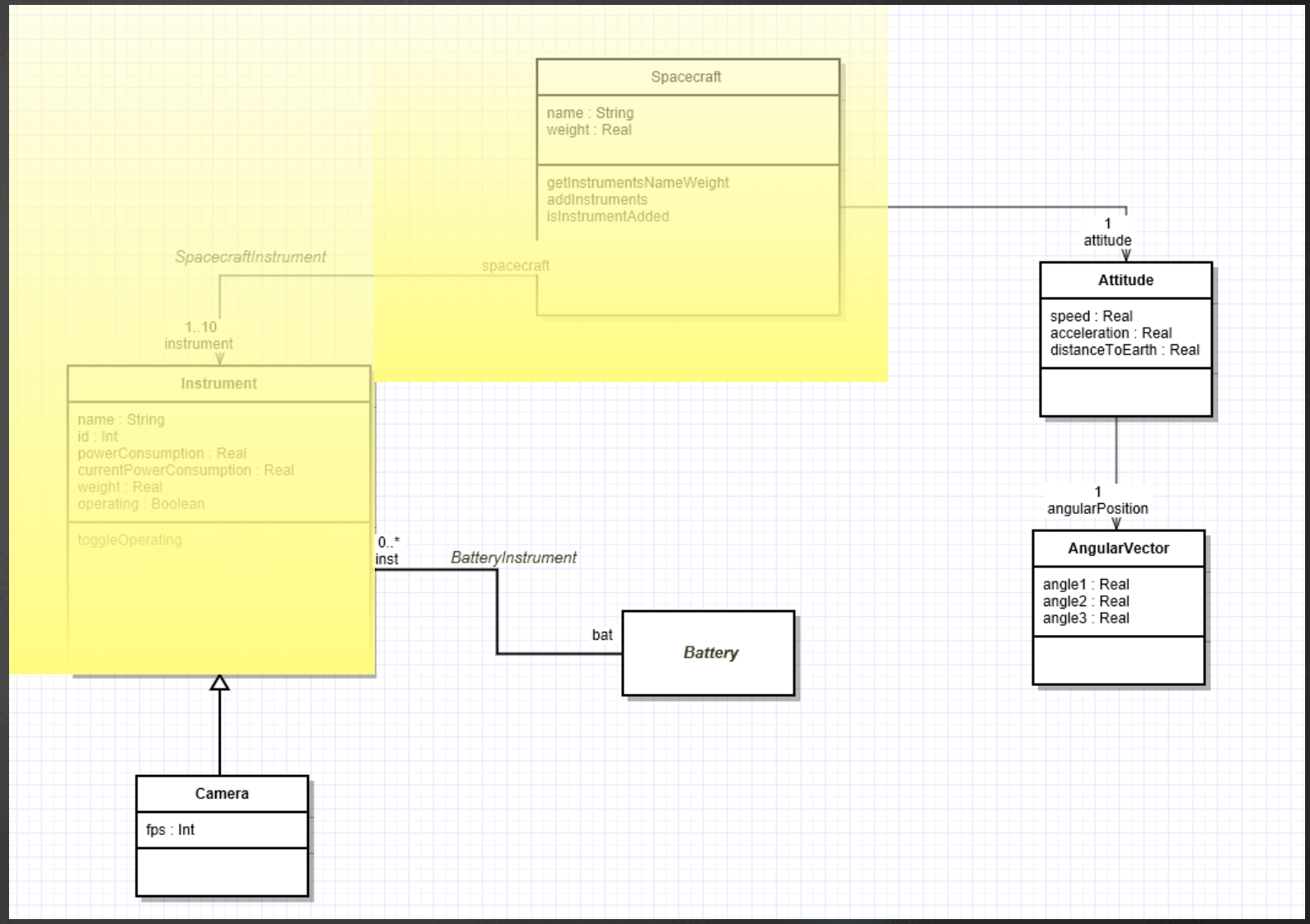
Spacecraft Example

```
@name("InstrumentClass")
@id("_GHSZ3432")
@doc("This class describes the basic instrument for a spacecraft.")

class Instrument {
  name : String
  id : Int
  @owner("Rahul")
  powerConsumption : Int = 1000
  @owner("Bjorn")
  weight : Real
  currentPowerConsumption : Int
  operating : Bool = false

  fun toggleOperating
    post(operating = !operating~)
  {
    operating := !operating
    if !operating then
      currentPowerConsumption := 0
    else
      currentPowerConsumption := powerConsumption
  }

  @name("OperatingPowerOfInstrument")
  @doc("The current power consumption of an instrument should either be 0,
        when it is turned off, or if it is on,
        it should be what the operating power is specified to be.")
  req OperatingPower:
    (!operating=> currentPowerConsumption = 0) &&
      (operating => currentPowerConsumption = powerConsumption)

  req idId: id >= 0

  req OperatingPower1:
      currentPowerConsumption = powerConsumption
}
```



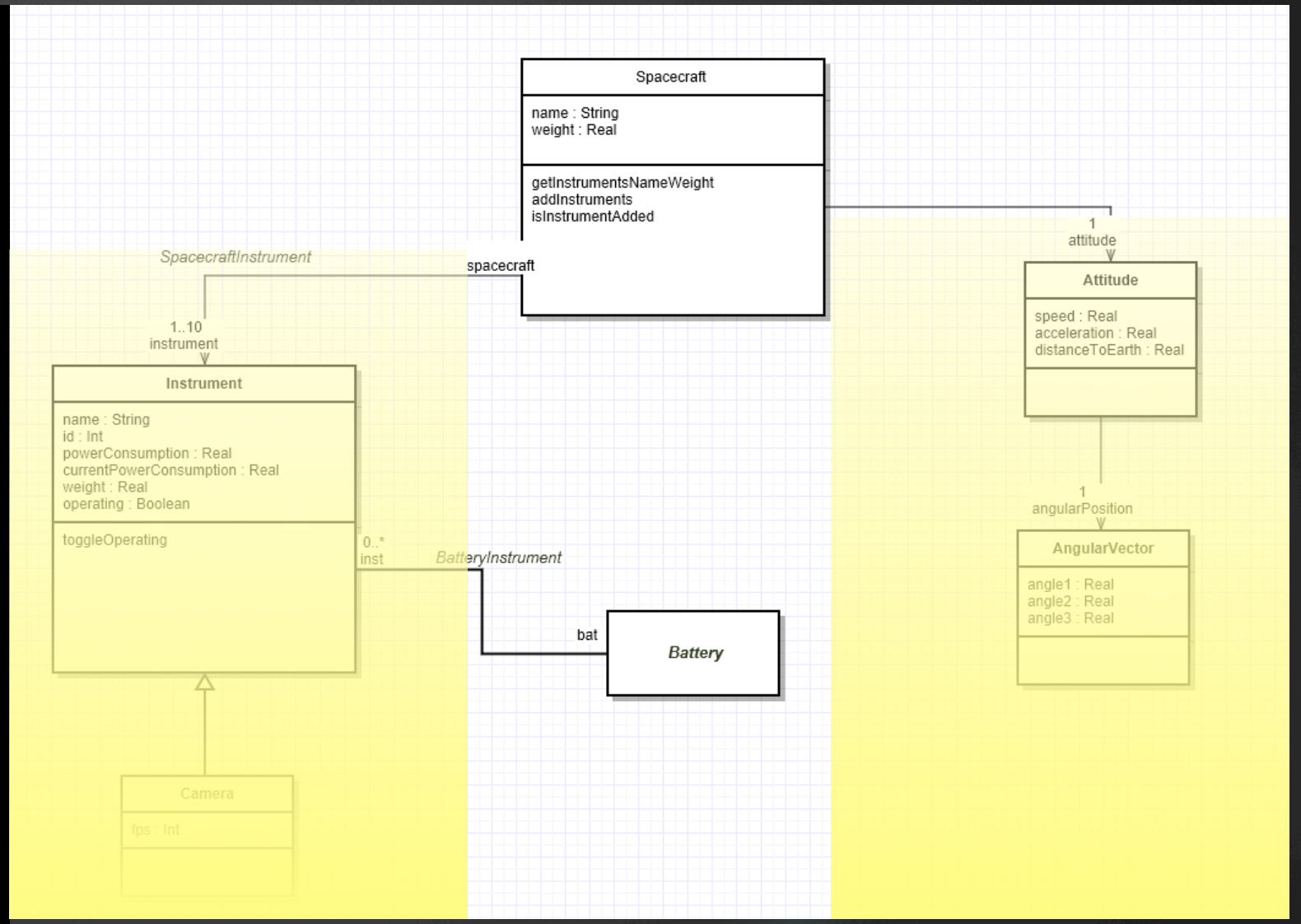
Spacecraft Example

```
class Camera extends Instrument {
  fps : Int
}

class Battery {}

class AngularVector {
  angle1 : Real
  angle2 : Real
  angle3 : Real
}

class Attitude {
  angularPosition : AngularVector
  speed : Real
  acceleration : Real
  distanceToEarth : Real
}


assoc BatteryInstrument {
  bat : Battery
  inst : Instrument [0,*]
}
```



Spacecraft Example

```
class Spacecraft {
  name : String
  attitude : Attitude
  weight : Real

  req EarthSafeDistance:
    attitude.distanceToEarth > 50000 &&
    attitude.distanceToEarth < 350000

  req notTooHeavy: totalWeight() <= maxWeight

  fun totalWeight : Real {
    instrument.collect(i -> i.weight).sum()
  }

  fun getInstrumentsNameWeight : Seq[String * Real]
    post $result.length() = instrument.size()
  {
    instrument.collect(i -> Tuple(i.name, i.weight)).toSeq()
  }

  fun addInstruments(insts : Set[Instrument])
    pre insts.size() <= 10
    pre forall i : insts . i !isin instrument
  {
    instrument := instrument union insts
  }

  fun isInstrumentAdded(instr : Instrument) : Bool {
    instr isin instrument
  }
}

assoc SpacecraftInstrument {
  spacecraft : Spacecraft
  part instrument: Instrument[1,10]
}
```



Extensible Keywords

```
class ModelElement{
  ...
}

class <view> View extends ModelElement {
  ...
}

view MyView {
  ...
}
```

You can use **view** instead of **class.**

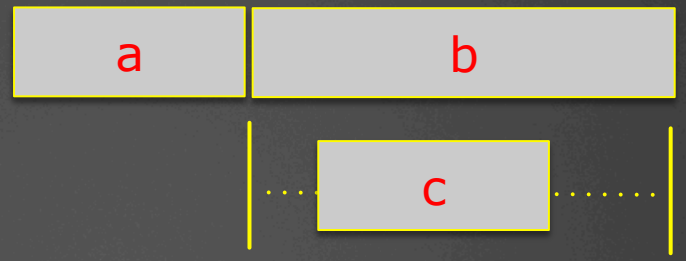
This is the same as writing

class MyView **extends** View {

Examples of K models and generated Z3 formulas

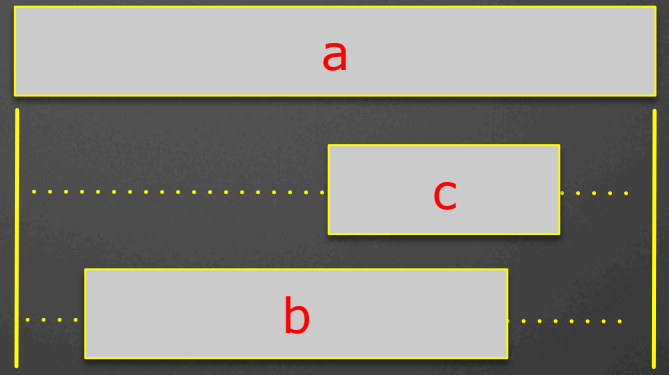
A simple planning scenario

Schedule 1

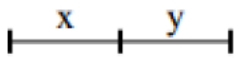
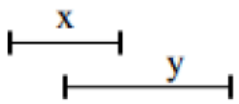
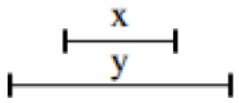
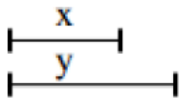
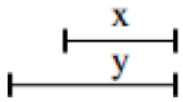
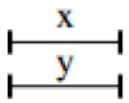


OR

Schedule 2



Allen’s interval logic

Relation	Symbol	Inverse	Meaning
x before y	b	bi	
x meets y	m	mi	
x overlaps y	o	oi	
x during y	d	di	
x starts y	s	si	
x finishes y	f	fi	
x equal y	eq	eq	

Further…

- Mapping from SysML diagrams (MagicDraw) to K and back
- Translation to Latex
- Translation to Z3, Mathematica, …
- Runtime environment
- And … determine relationship to existing programming language(s)
  - Python is used by even system engineers (solar system model in Python exists)
  - Java is used all over the place, but mostly by software engineers
  - Scala would be the closest match from a language point of view

More questions