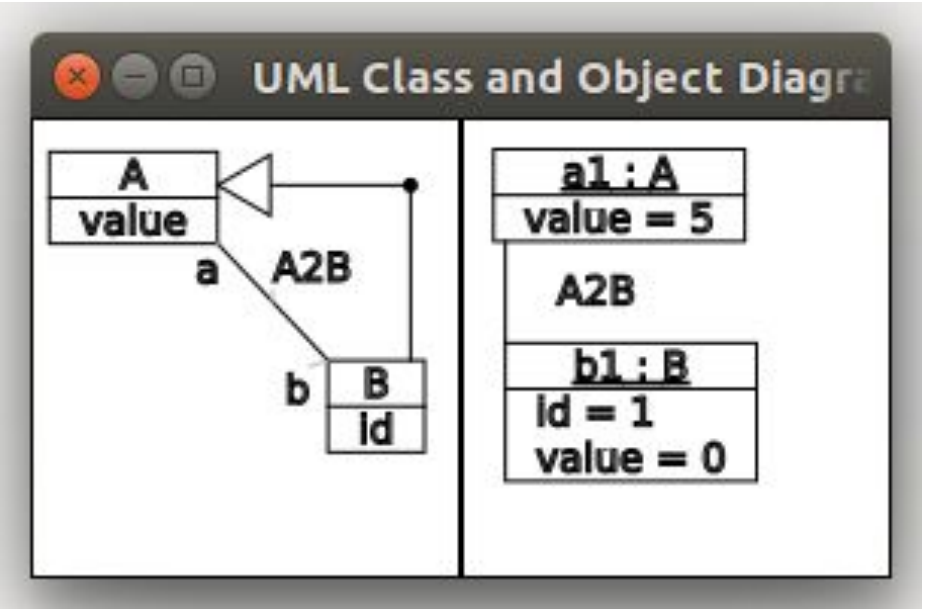
Intensional View Definition with Constrained Incremental Transformation Rules

Théo Le Calvar[1,2], *Frédéric Jouault*[2], Fabien Chhel[2], Frédéric Saubion[1], Mickael Clavreul[2]

[1] LERIA, University of Angers, Angers, France
{firstname.lastname}@univ-angers.fr
[2] ERIS Team, Groupe ESEO, Angers, France
{firstname.lastname}@eseo.fr

Example Overview: UML Class and Object Diagrams

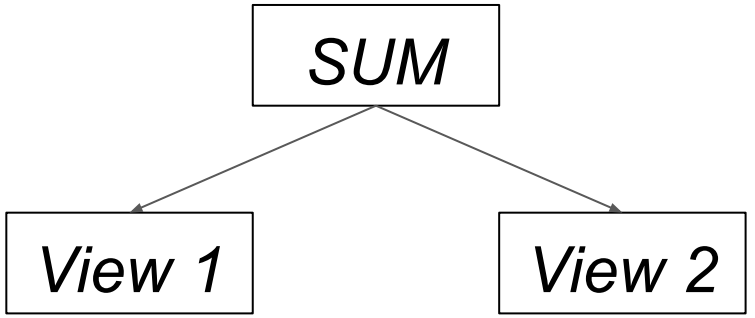


- A single underlying UML model
- Two visual JavaFX[1] views
  - Class diagram
  - Object diagram
- Changing either view changes the UML model
  - Which may also result in changes to the other view

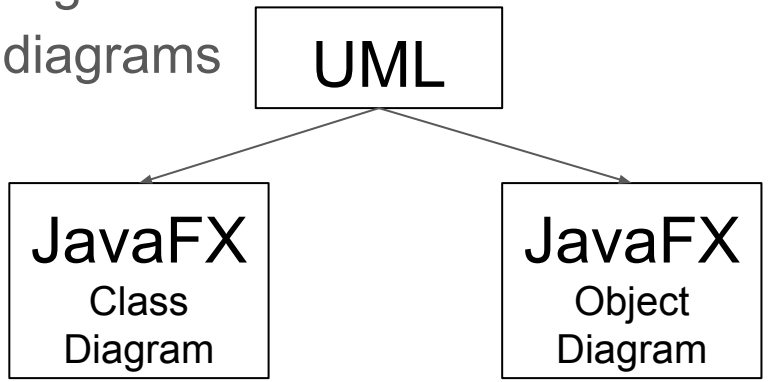
[1] JavaFX provides vectorial graphics support for Java programs

Synchronization: With Incremental Projective Views

- Incremental transformations
  - Partially bidirectional

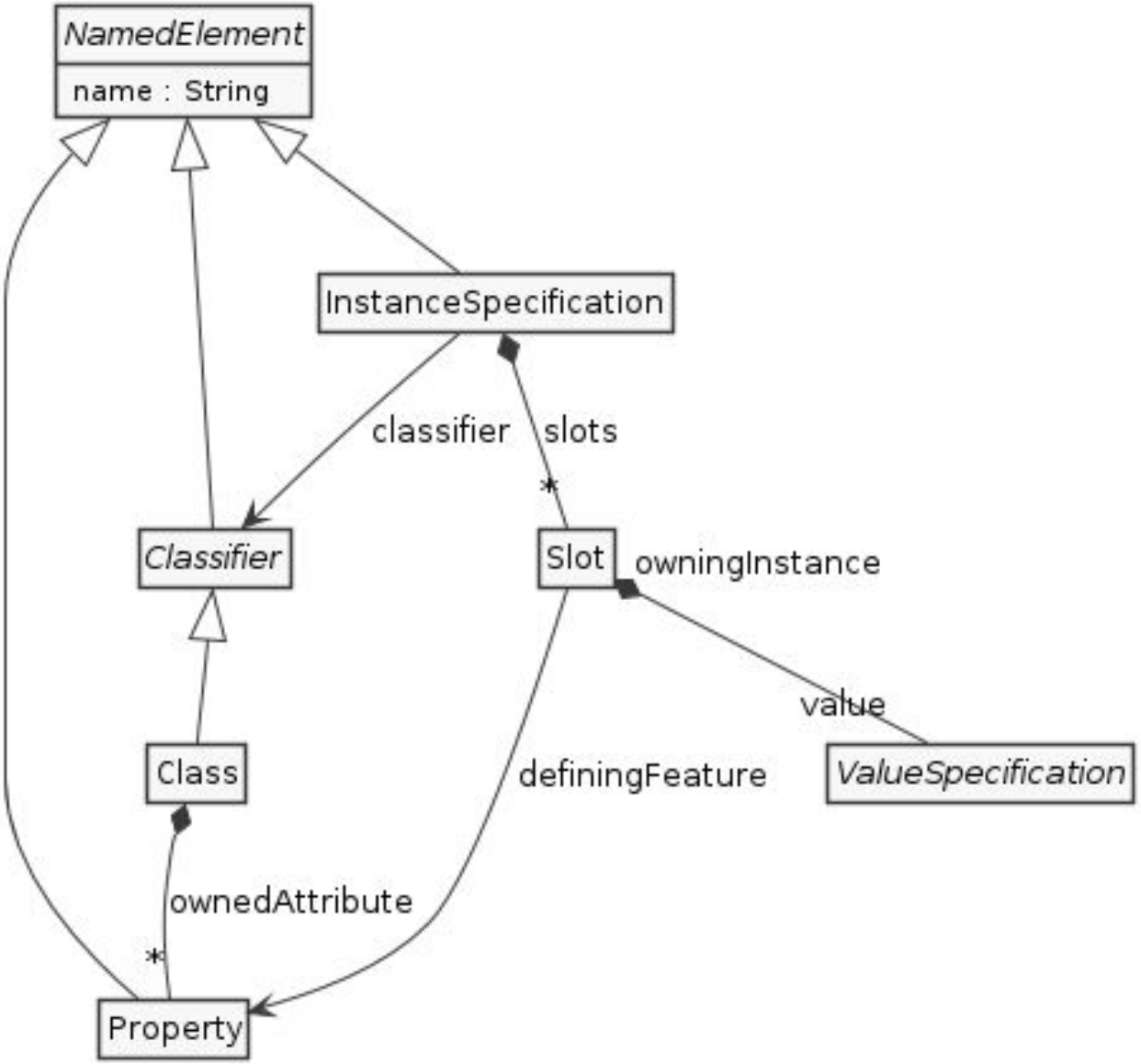


- For the example:
  - One ATOL[1] transformation for class diagrams
  - One ATOL[1] transformation for object diagrams

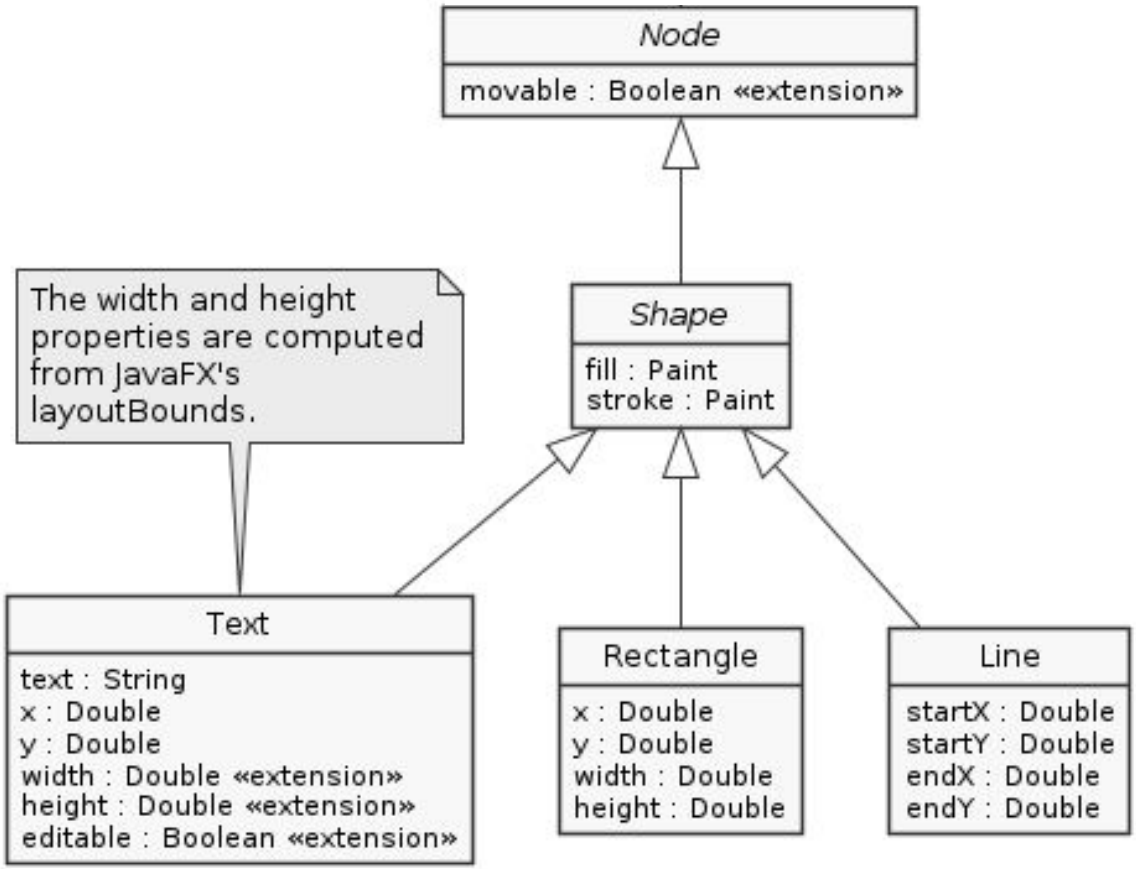


[1] ATOL is a new ATL compiler supporting incremental transformations

Example: Simplified UML Metamodel

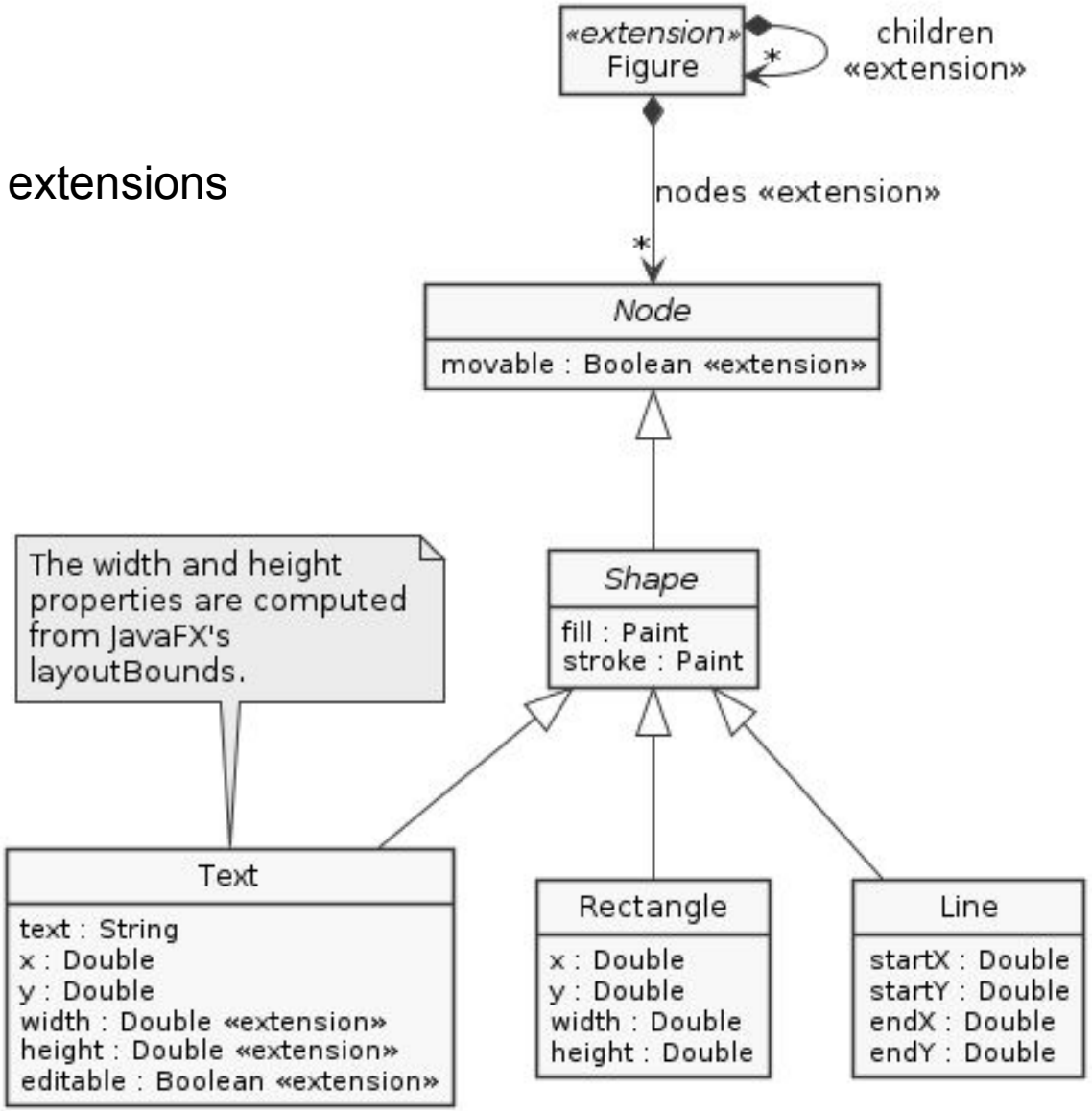


Example: Simplified Vectorial Drawing Metamodel (JavaFX)



Example: Simplified Vectorial Drawing Metamodel (JavaFX)

With some extensions



```
unique lazy rule Class {
  from
    s : UML!Class
  to
    t : JFX!Figure (
      nodes <- Sequence {r, txt, l},
      children <-
        s.ownedAttribute->collect(e |
          thisModule.Property(e).t
        )
    ),
    r : JFX!Rectangle (
      movable <- true,
      fill <- thisModule.transparentWhite,
      stroke <- thisModule.opaqueBlack
    ),
    txt : JFX!Text (
      text <- s.name,
      fill <- thisModule.transparentWhite,
      stroke <- thisModule.opaqueBlack,
      editable <- true,
      movable <- true
    ),
    l : JFX!Line (
      stroke <- thisModule.opaqueBlack
    )
}
```

Example Projection Rule

- For each Class, generate
  - A Rectangle outline
  - A Text for its name
  - A Line as separator
- For each Property, generate
  - A Text for its name
- For both, generate Figures
  - That connect all created elements together in a tree structure
  - That make it possible to retrieve all target elements necessary to draw a given source element

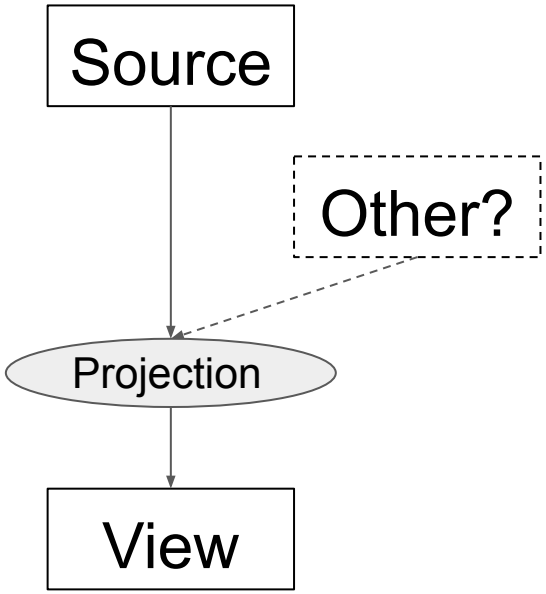
```
unique lazy rule Class {
  from
    s : UML!Class
  to
    t : JFX!Figure (
      nodes <- Sequence {r, txt, l},
      children <-
        s.ownedAttribute->collect(e |
          thisModule.Property(e).t
        )
unique lazy rule Property {
  from
    s : UML!Property
  to
    t : JFX!Figure (
      nodes <- Sequence {txt},
      children <- Sequence {}
    ),
    txt : JFX!Text (
      text <- s.name,
      fill <- thisModule.transparentWhite,
      stroke <- thisModule.opaqueBlack,
      editable <- true,
      movable <- true
    )
}
```

Example Projection Rule

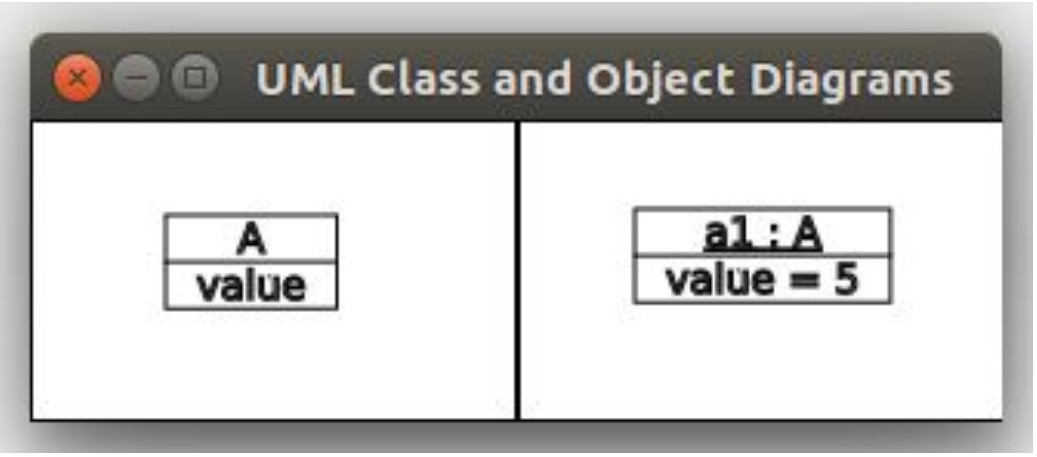
- For each Class, generate
  - A Rectangle outline
  - A Text for its name
  - A Line as separator
- For each Property, generate
  - A Text for its name
- For both, generate Figures
  - That connect all created elements together in a tree structure
  - That make it possible to retrieve all target elements necessary to draw a given source element

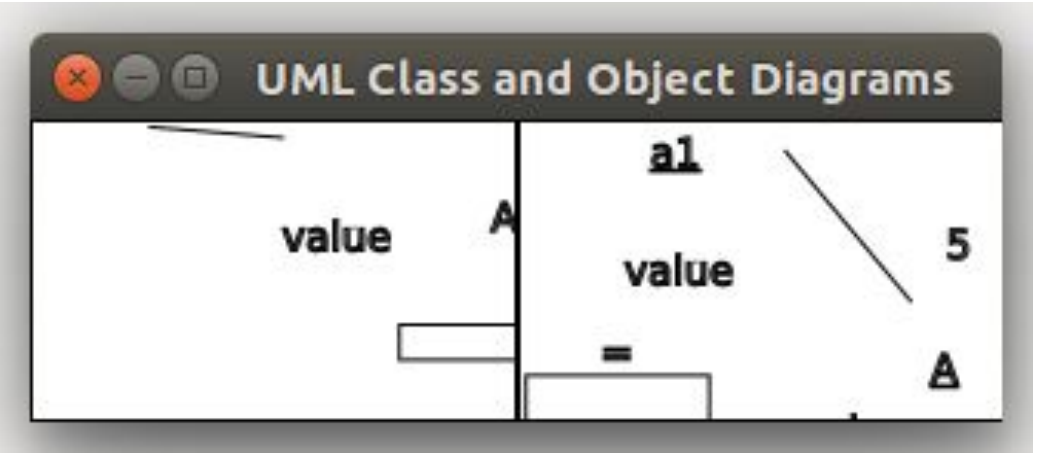
Problem

- Some views contain information not found in their source models
- Such information may
  - Already exist somewhere and be given as an additional projection transformation input
  - **Be unavailable, but follow some pattern**
    - i.e., not all possible values are valid

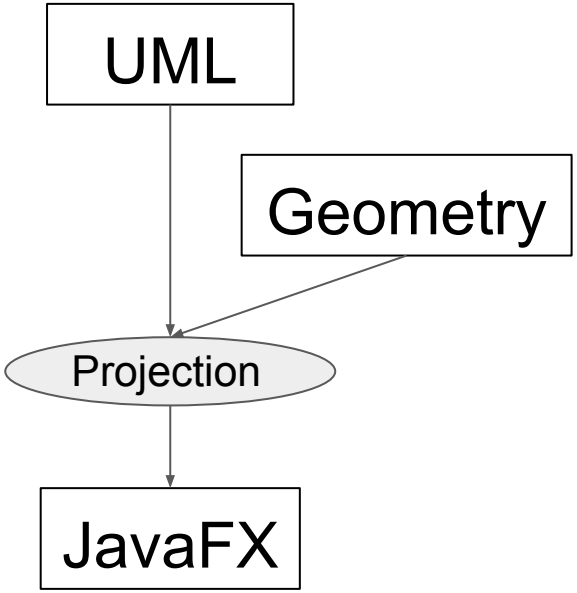


Example: What About Geometric Information?

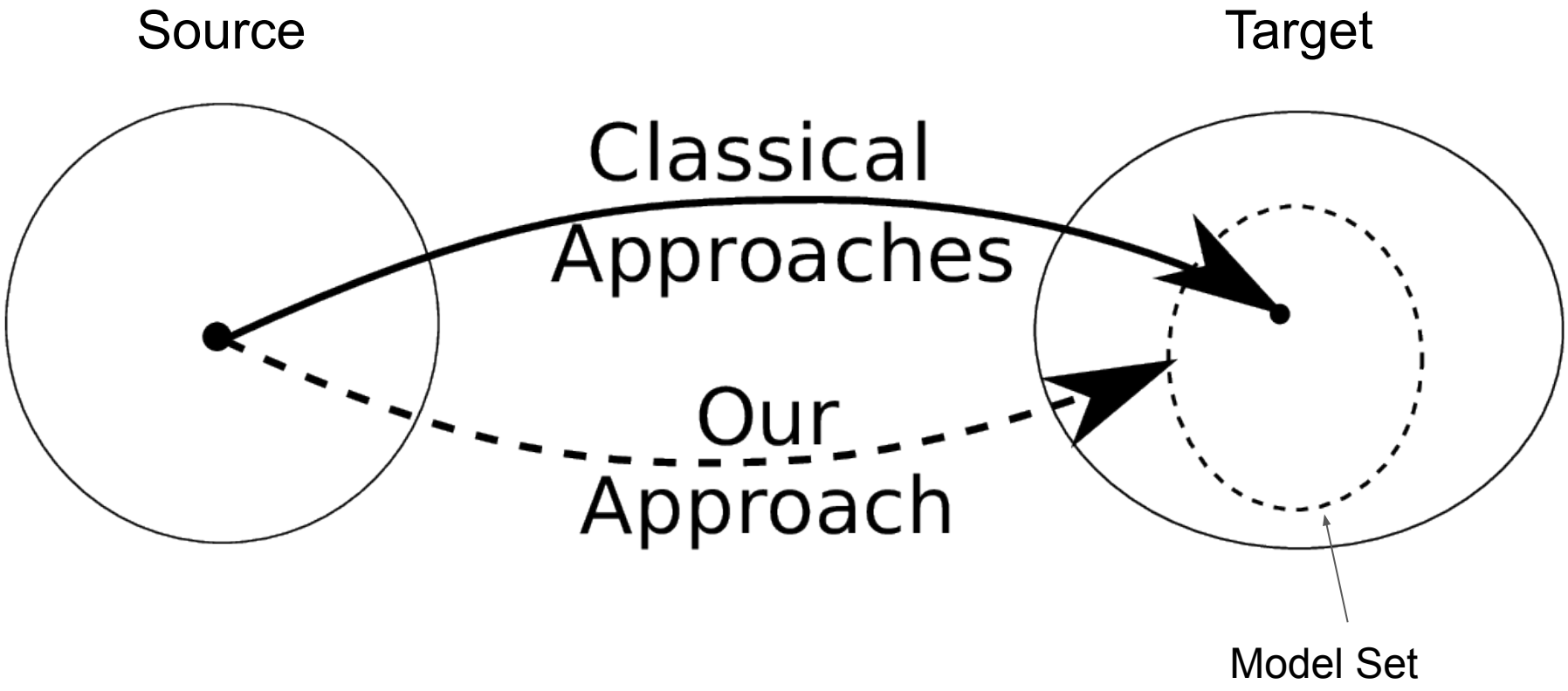




- Geometric information
  - micro-layout
    - follows strict rules
  - macro-layout
    - may be user specified



Approach: Intensional Target Model

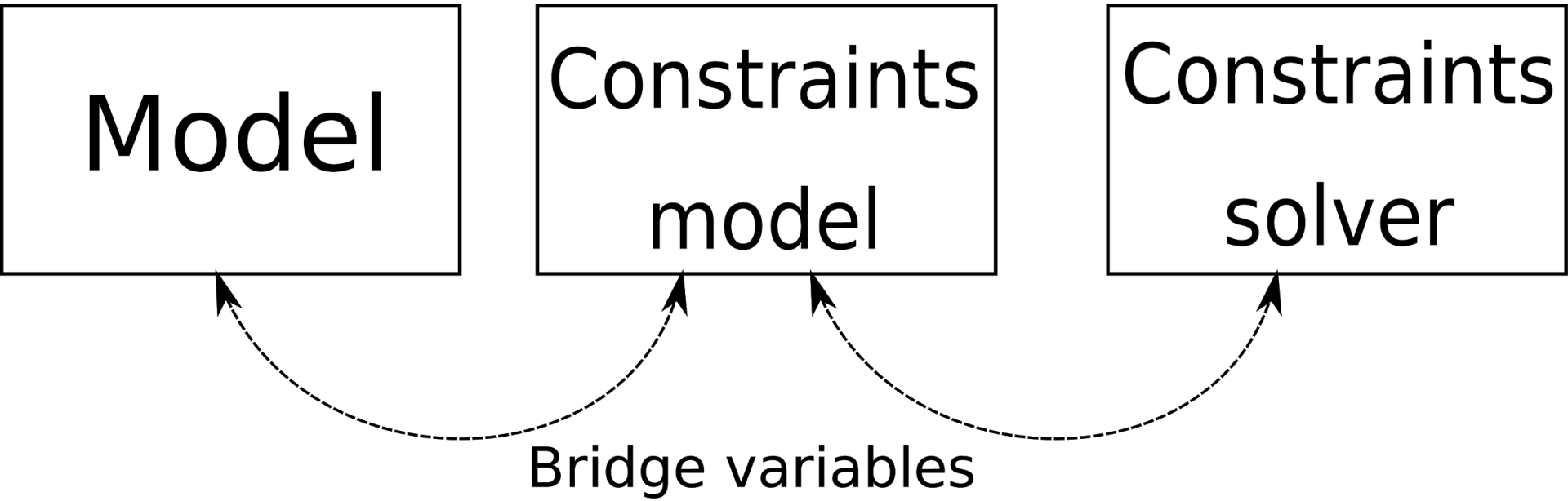


Approach: Constraint-based Intensional Model Definition

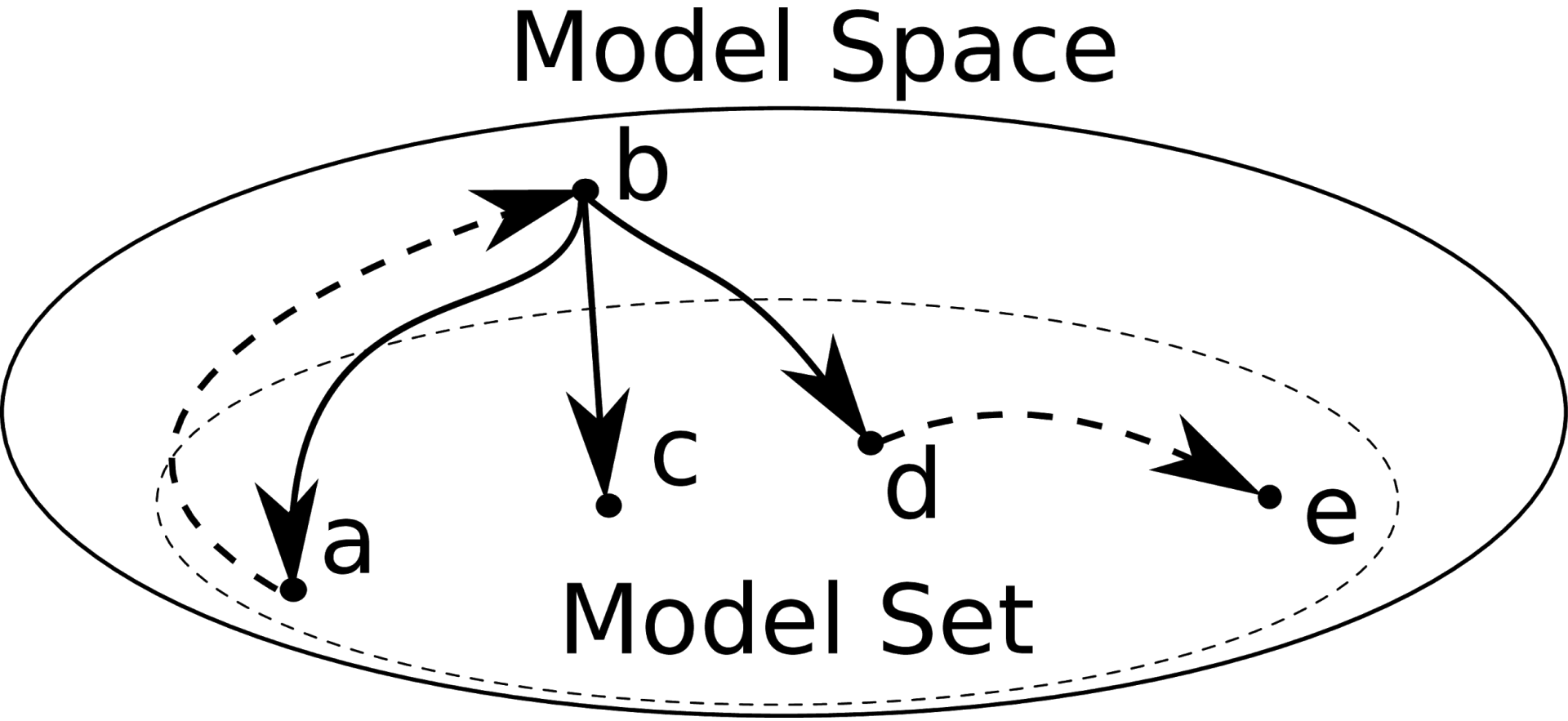
- Recipe
  - Take a *partially* instantiated model
    - i.e., with model element properties having no specific values
  - Add constraints between these properties
  - Use a solver to give them valid values
- The solver's capabilities delimit what applications are possible, for instance, it may
  - Be fast enough for live interactions, or not
  - Only support linear constraints, or more complex ones

Connecting Models to Solvers

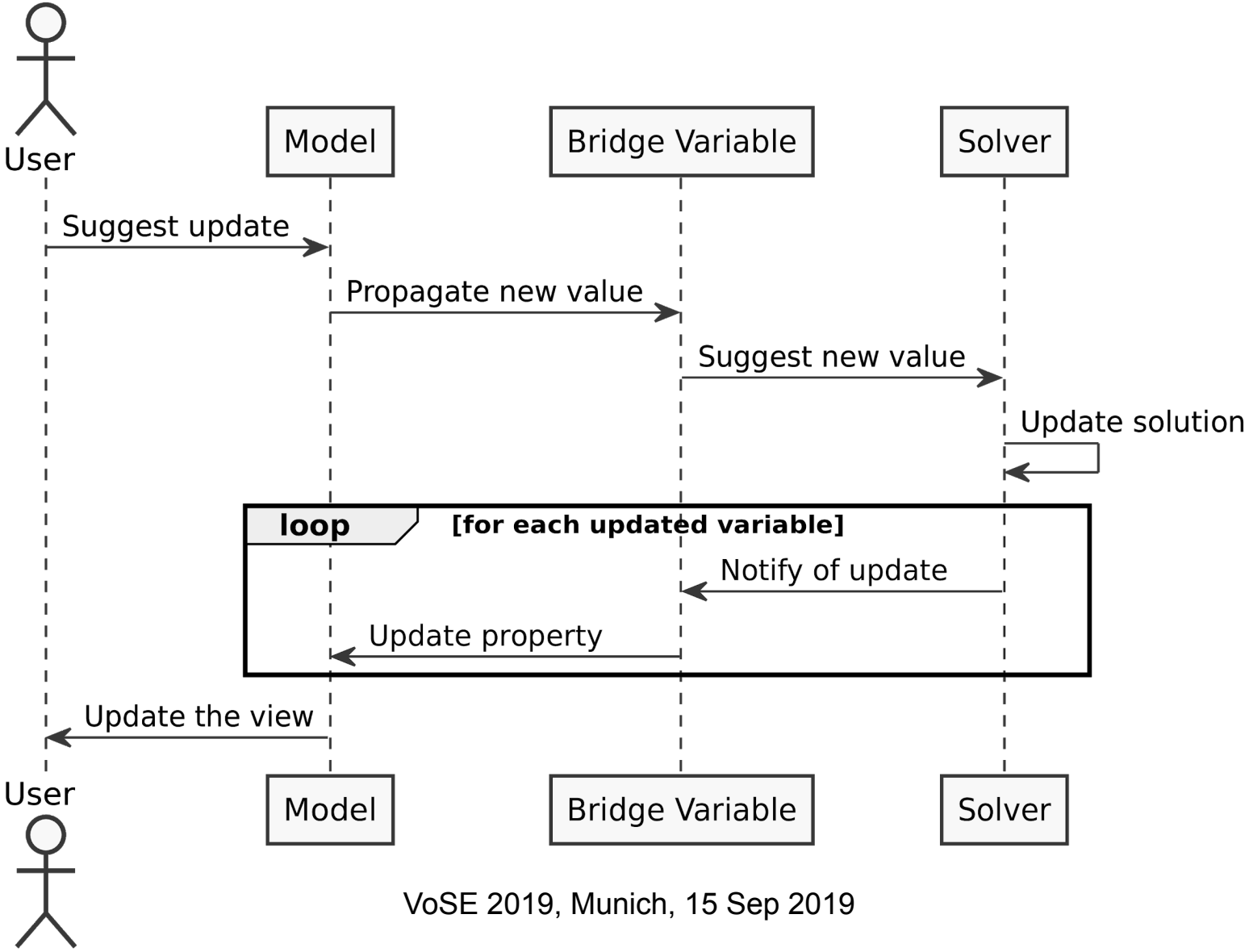
- Model element properties are bidirectionally bridged to solver decision variables



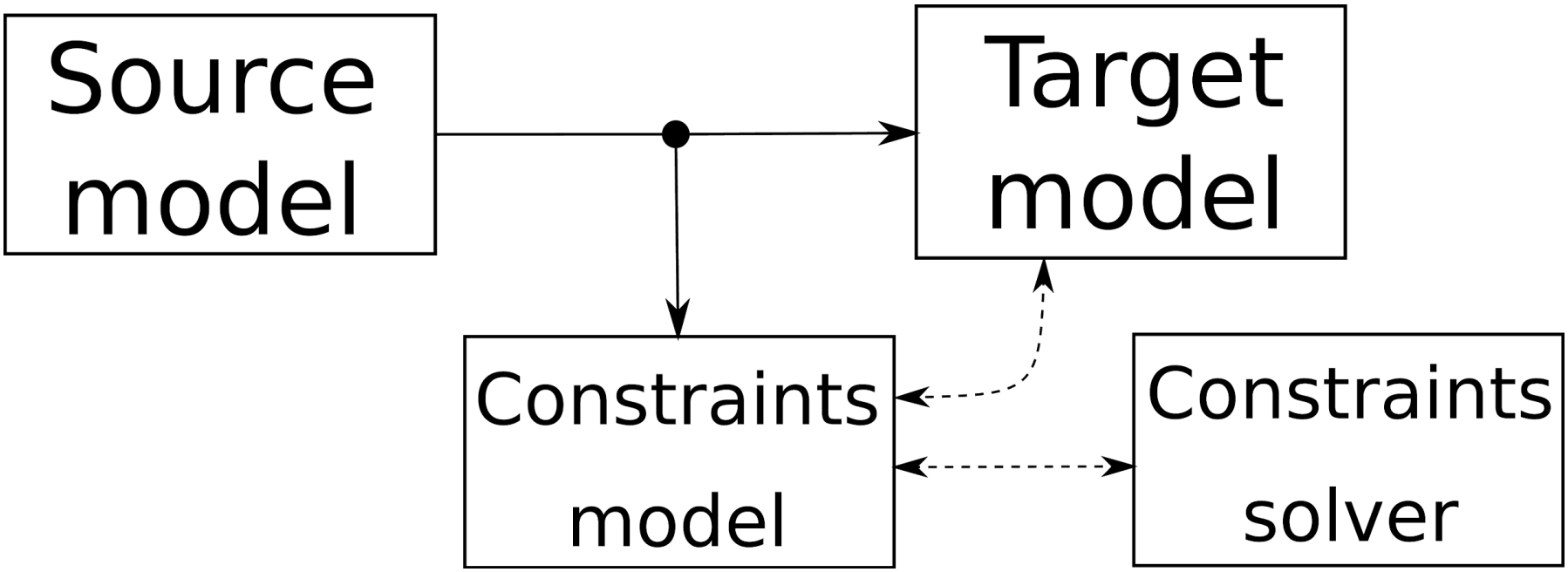
Model Set Exploration



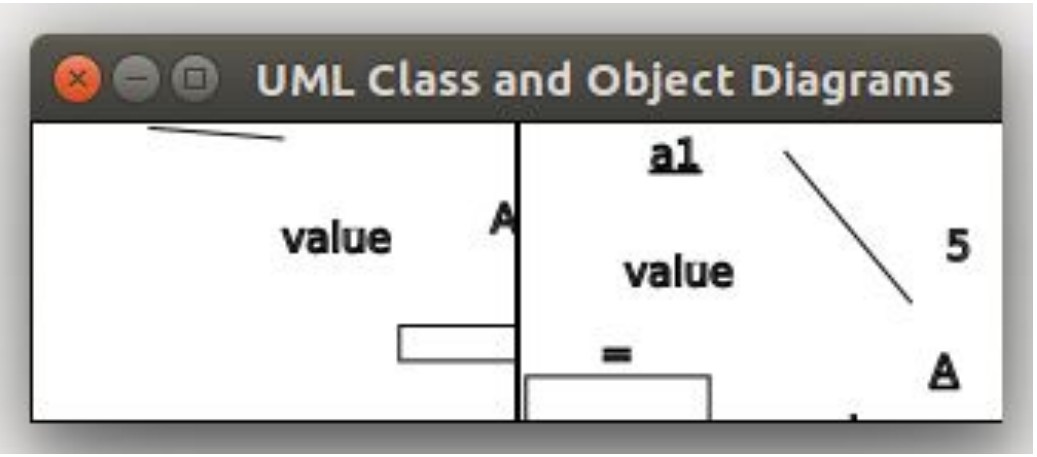
Overview of Model Set Exploration

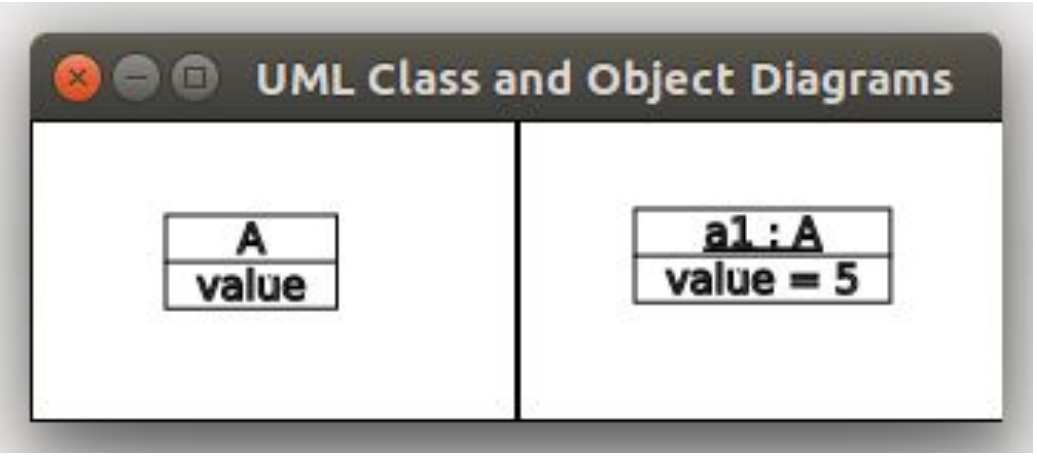


Combining Incrementality & Intensionality



Example: Class Constraints





- Avoid moving rectangle and text
- Minimize rectangle size
- Prevent rectangle from going above or left of canvas
- Center text horizontally
- Stick text to rectangle top
- Make rectangle larger than text + an horizontal margin
- Make the line
  - Go from left to right of rectangle
  - Be horizontal
  - Be just below the text

Example: Class Constraints

```
unique lazy rule Class {
  from
    s : UML!Class
  to
    t : JFX!Figure(
        ...,
        constraints <- Sequence {constraints}
    ),
    r : JFX!Rectangle (...),
    txt : JFX!Text (...),
    l : JFX!Line (...),
    constraints : Constraints!ConstraintGroup (
      solver <- 'cassowary',
      constraints <-
        let txtCenterX : Real =
            txt.x + txt.width.toConstant() / 2
        in
        let rCenterX : Real =
            r.x + r.width / 2
        in
        Sequence {
          r.x.stay('MEDIUM'), r.y.stay('MEDIUM'),
          txt.x.stay('MEDIUM'), txt.y.stay('MEDIUM'),

          r.width = 0 -- STRONG
          ,r.height = 0 -- STRONG
          ,r.x >= 0, r.y >= 0,

          rCenterX = txtCenterX,
          r.y = txt.y,

          r.width >= txt.width.toConstant() +
              thisModule.MARGIN * 2,
          r.height >= txt.height.toConstant(),

          l.startX = r.x, l.endX = r.x + r.width,
          l.startY = l.endY,
          l.startY = txt.y + txt.height.toConstant()
        }
    )
}
```

- Avoid moving rectangle and text
- Minimize rectangle size
- Prevent rectangle from going above or left of canvas
- Center text horizontally
- Stick text to rectangle top
- Make rectangle larger than text + an horizontal margin
- Make the line
  - Go from left to right of rectangle
  - Be horizontal
  - Be just below the text

Conclusion

- The presented approach makes it possible to
  - Declaratively define incremental (partially bidirectional) intensional views
  - Explore these intensional models
- It has been illustrated on a visual view example
  - But is applicable to other kinds of views (e.g., schedule)
- Can be used with multiple solvers at the same time
  - When there are no dependency cycles between bridge variables
- Perspectives:
  - Domain specific abstractions (e.g., geometric abstractions)
  - Possible example extensions: macro-layout constraints