IWST — *9 July 2026*

MicroUML: Building a UML DSL Without a Compiler

Oleksandr ZAITSEV
CIRAD, UMR SENS
oleksandr.zaitsev@cirad.fr

Tomohiro ODA
Software Research Associates
tomohiro@sra.co.jp

Stéphane DUCASSE
Inria, Evref, CRIStAL
stephane.ducasse@inria.fr

Cool idea from ESUG 2025

MicroUML began as a spontaneous experiment during ESUG

Tomo’s presentation of Micromaid

↓

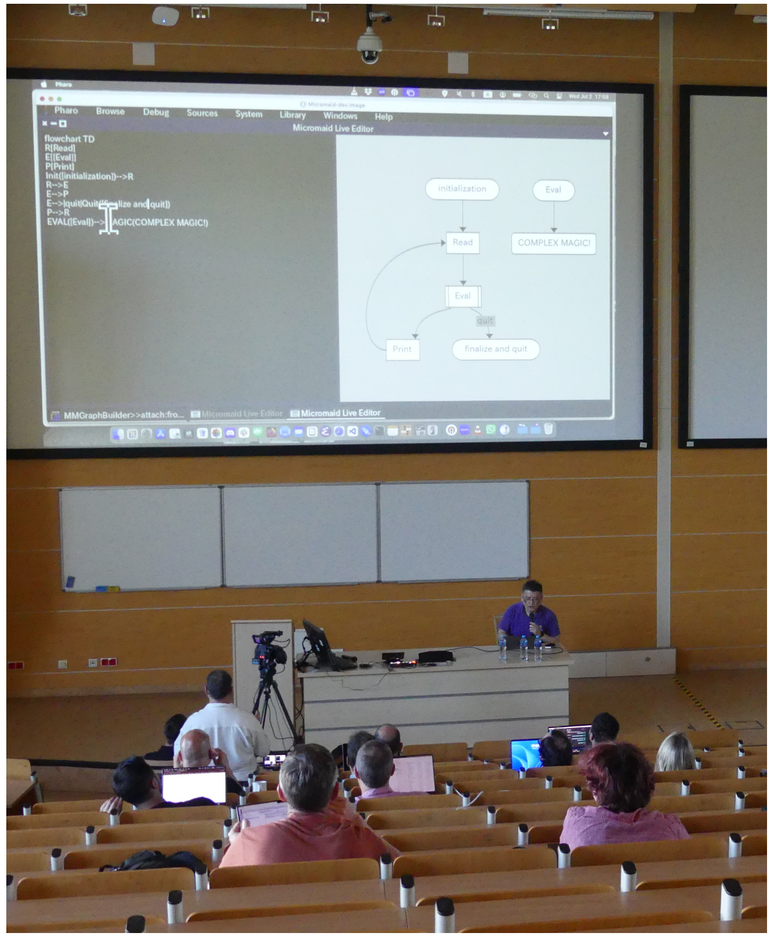
Can we build UML DSL using only standard Pharo syntax?

↓

Pair-programming during the coffee break

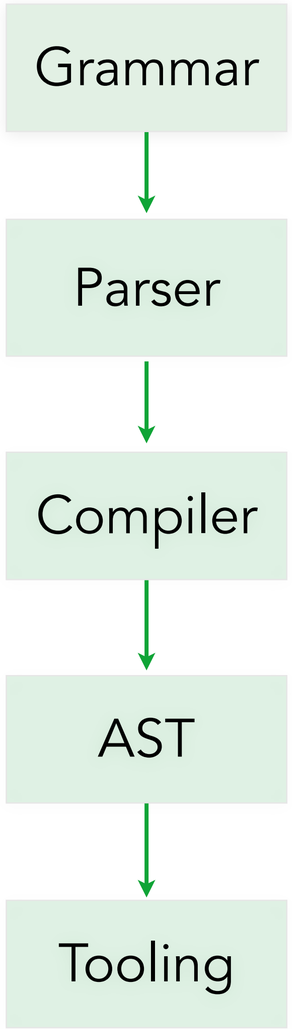
↓

First prototype of MicroUML



Challenges of Traditional DSLs

Traditional DSLs require building and maintaining an entire language infrastructure



Challenge 1:

- ✗ Hard to implement
- ✗ Hard to maintain
- ✗ Hard to evolve

Challenge 2:

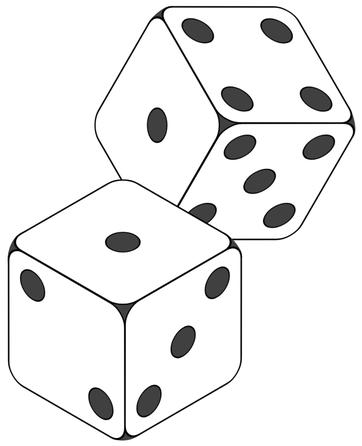
DSL artifacts ≠ Runtime objects

⟶ models are disconnected from the live environment

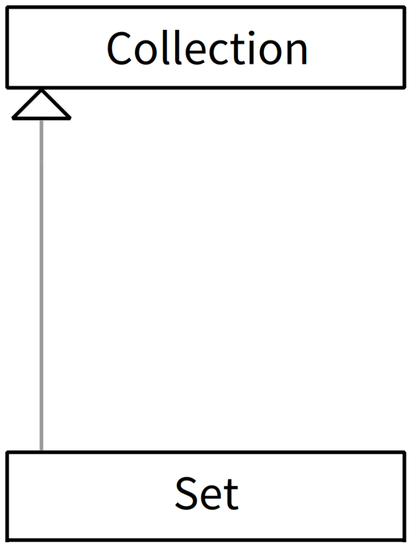
From the Dice Exercise to MicroUML

The same message-based technique scales from a simple educational DSL to a practical modeling language

```
2 D20 + 3 D10
```



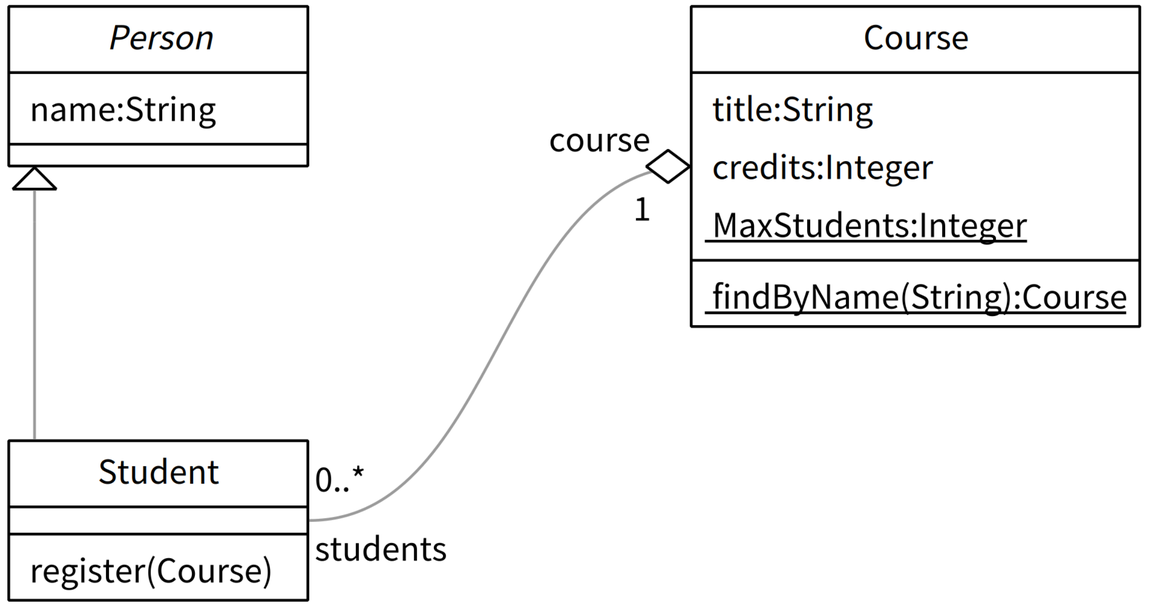
```
Set --|> Collection
```



MicroUML by example

A complete UML class diagram can be expressed as ordinary Pharo code

```
#Person % #abstract
   - #name @ String
===
#Student --|> #Person
   > register ~ #(Course)
===
#Course
   - #title @ String
   - #credits @ Integer
   |- #MaxStudents @ Integer
   |> #findByName ~ #(String) @ Course
   <>--- #Student
       @ (‘course’ -> ‘students’)
       %< ‘1’
       %> ‘0..*’
```



Classes, inheritance and traits

Core UML concepts are represented using simple Pharo message sends

Class name

```
#Class
```

Modifiers

```
#Class % #abstract
```

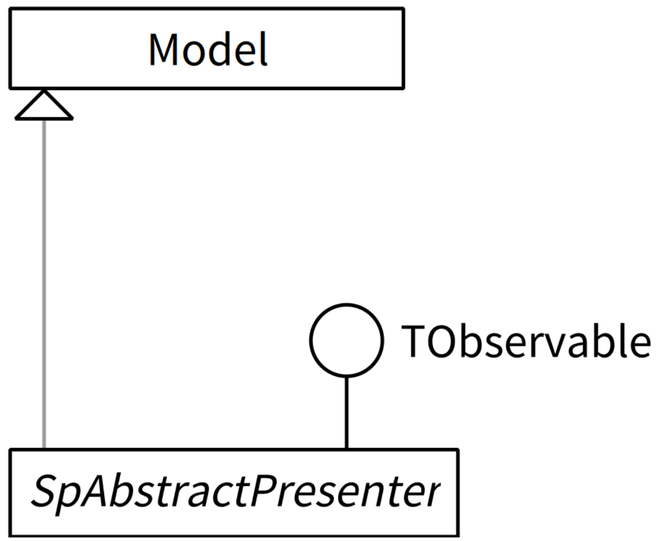
Inheritance

```
#Class --|> #Superclass
```

Traits

```
#Class --@ #Trait
```

```
SpAbstractPresenter % #abstract
    --|> Model
    --@ TObservable
```



Attributes

Attributes are declared using concise message chains that closely mirror UML notation

Variable name

```
– #instanceSideVar
|– #classSideVar
```

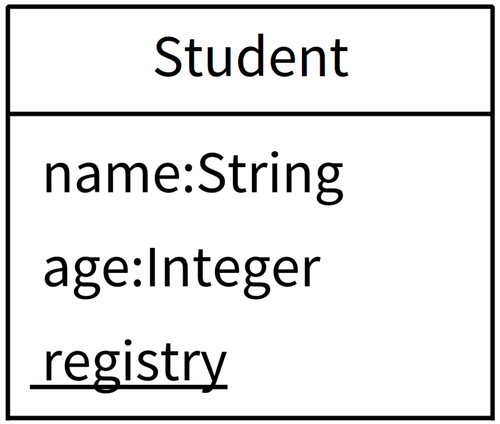
Type

```
– #age @ Integer
```

Access modifiers

```
– #id % #private
```

```
#Student
    – #name @ String
    – #age @ Integer
    |– #registry
```



Methods

Method signatures naturally express parameters and return types using ordinary Pharo syntax

Method name

```
> #instanceSideMethod
|> #classSideMethod
```

Arguments

```
> #collect ~ #(Block)
```

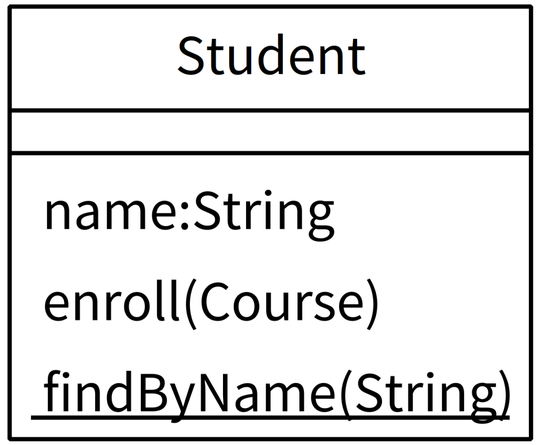
Return type

```
> #size @ Integer
```

Access modifiers

```
> #id % #private
```

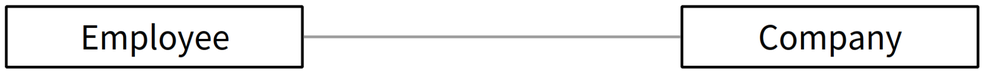
```
#Student
    > #name @ String
    > #enroll ~ #(Course)
    |> #findByName ~ #(String)
```



Associations

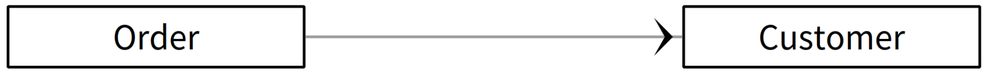
Association

```
Employee -- Company
```



Directed association

```
Order => Customer
```



Aggregation

```
Library <>-- Book
```

Directed aggregation

```
Library <>=> Book
```

Composition

```
Car *-- Wheel
```

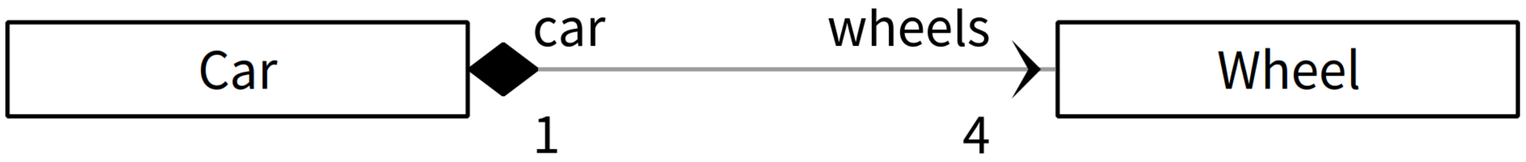
Directed composition

```
Car *=> Wheel
```

Associations: Roles & Multiplicities

Roles and multiplicities are expressed through chained messages that closely resemble UML

```
Car *=> Wheel
    @ (#car -> #wheels)
    %< 1
    %> 4
```



Putting it all together

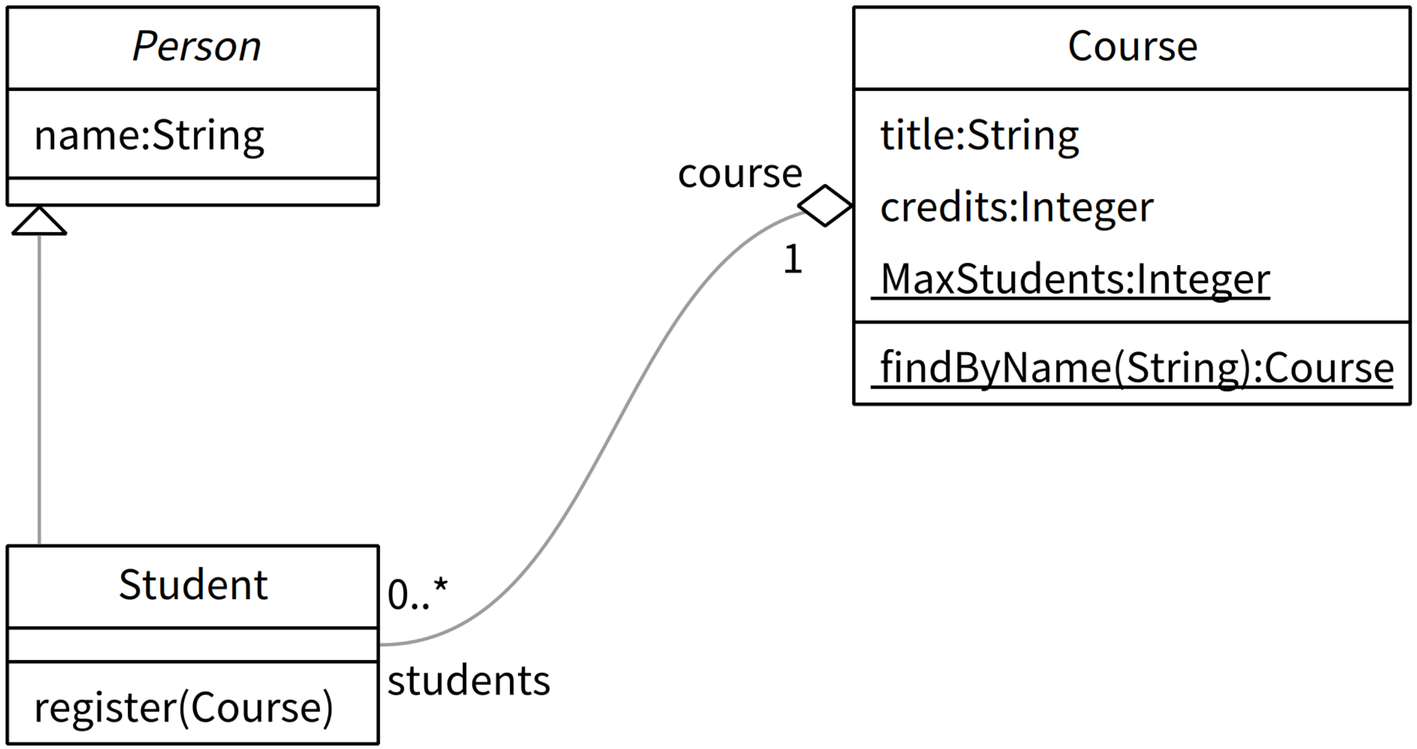
MicroUML covers a substantial subset of UML class diagrams with a compact syntax

```
#Person % #abstract
   - #name @ String
===
#Student --|> #Person
   > register ~ #(Course)
===
```

```
#Course
   - #title @ String
   - #credits @ Integer
   |- #MaxStudents @ Integer
   |> #findByName ~ #(String) @ Course
   <>-- #Student
       @ (‘course’ -> ‘students’)
       %< ‘1’
       %> ‘0..*’
```

Putting it all together

MicroUML covers a substantial subset of UML class diagrams with a compact syntax



Everything is a chain of message sends

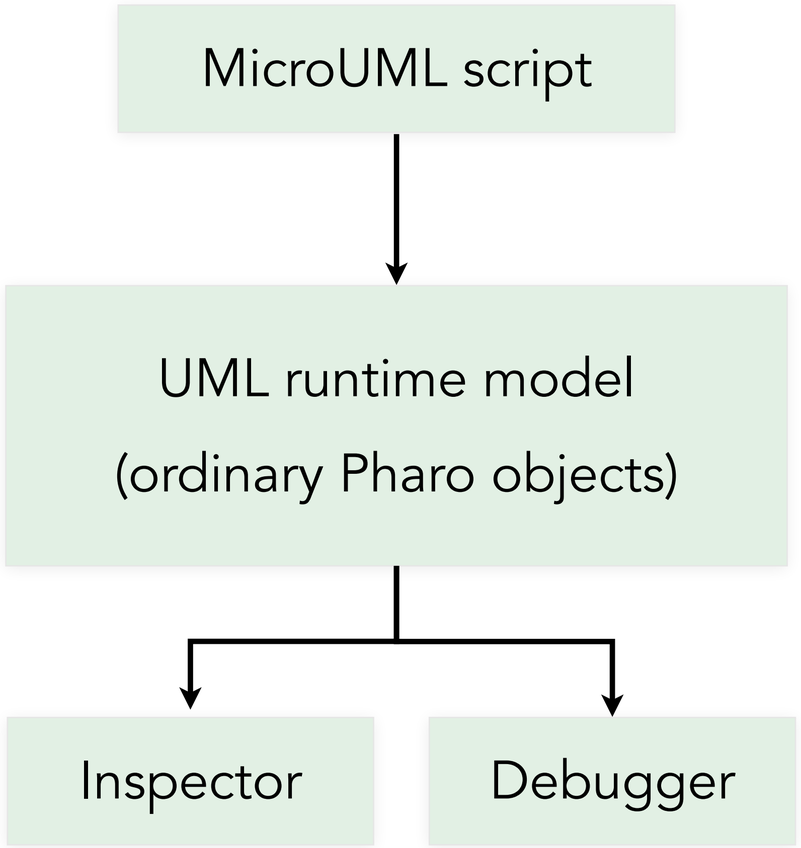
A rich UML DSL can be expressed entirely using Pharo's message syntax

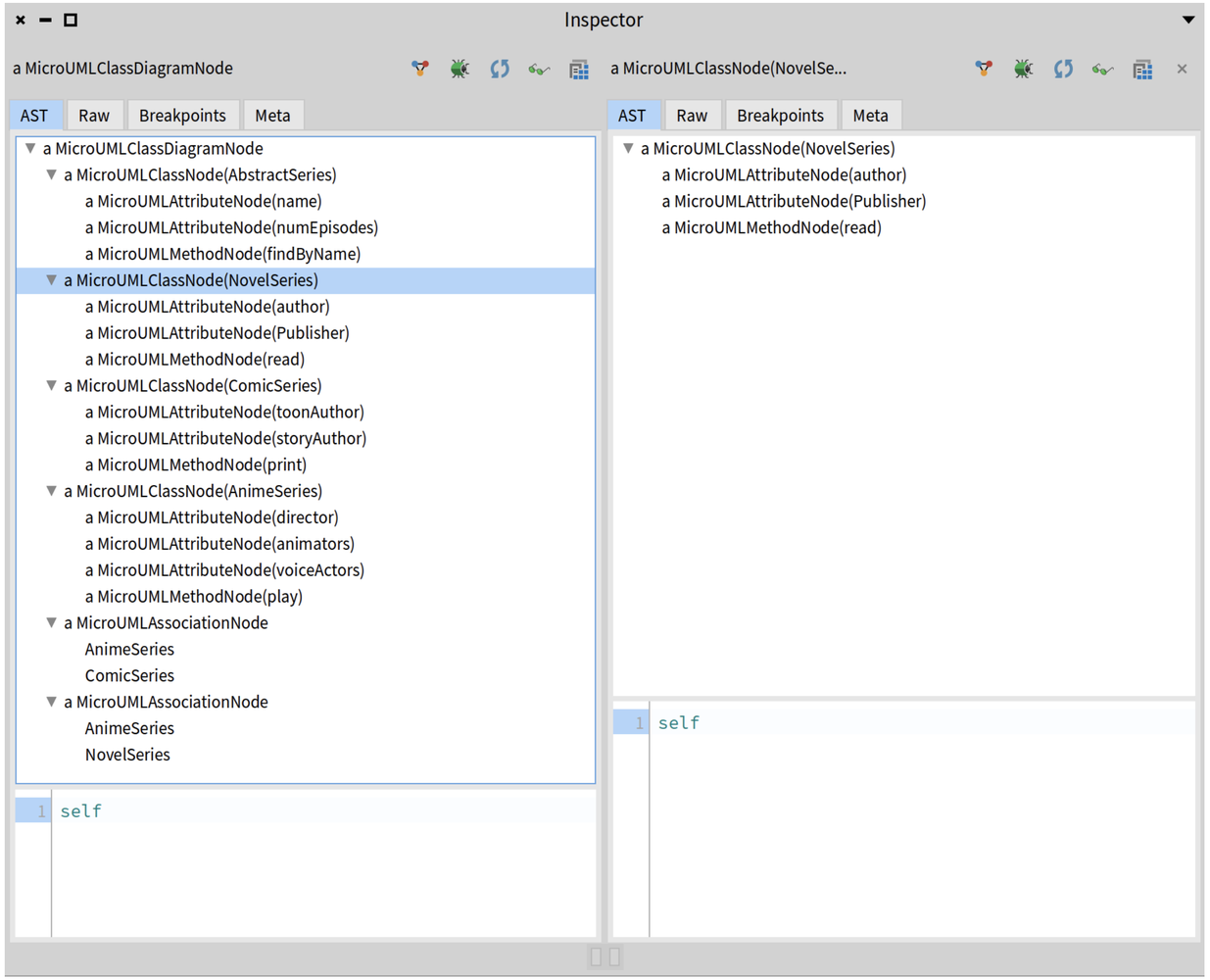
```
#Person % #abstract
    - #name @ String
===
#Student --|> #Person
    > register ~ #(Course)
===
```

```
#Course
    - #title @ String
    - #credits @ Integer
    |- #MaxStudents @ Integer
    |> #findByName ~ #(String) @ Course
    <>-- #Student
        @ (‘course’ -> ‘students’)
        %< ‘1’
        %> ‘0..*’
```

Models are runtime objects

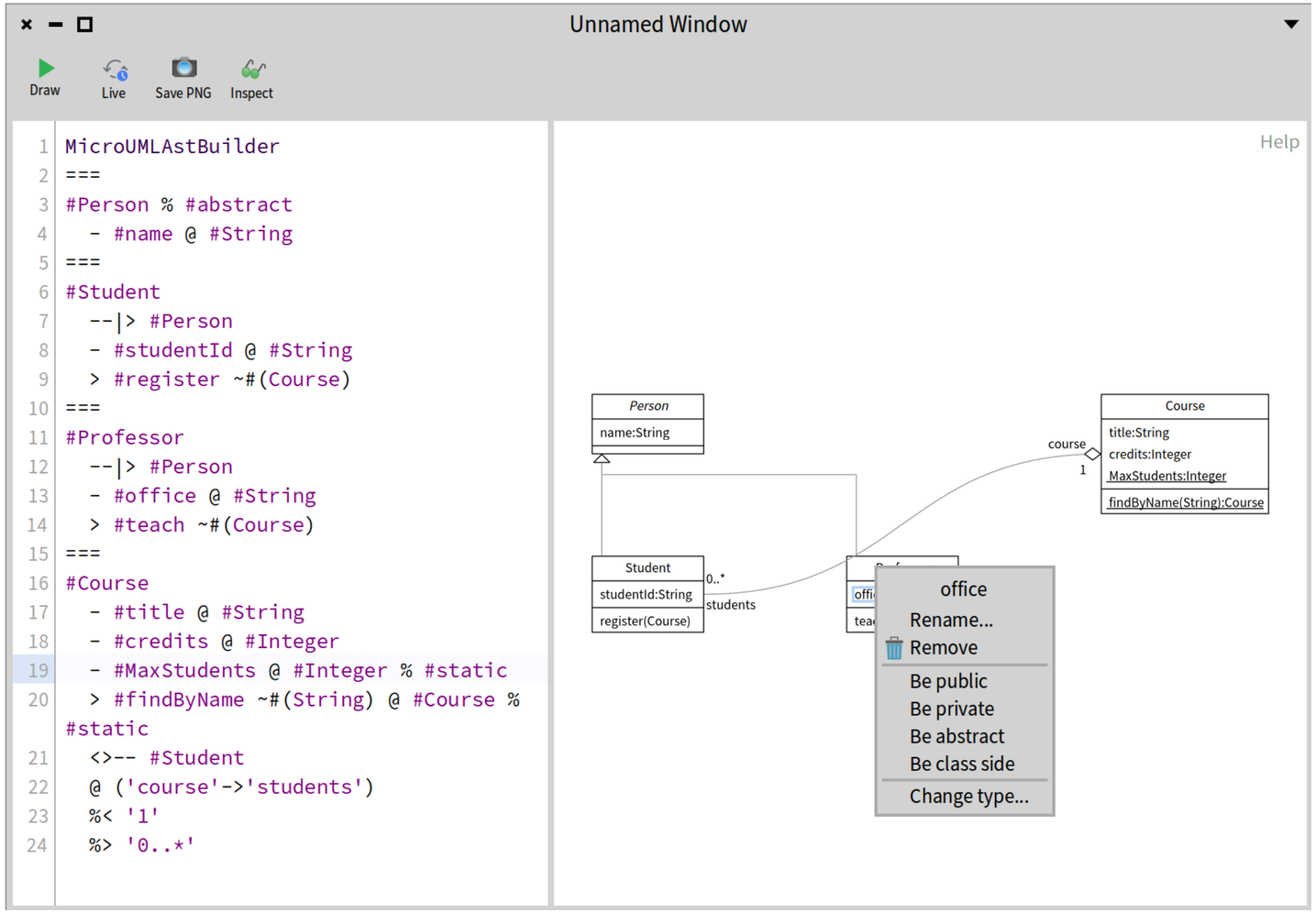
Executing a MicroUML script produces an inspectable object model





Tool support

We implemented some tools to support the UML workflow



Live interactive editor

- Live editor & package browser
- Interactive visualization
- PNG export

Reverse engineering & Code generation

Models and code can be transformed into each other, enabling a continuous development workflow.

Generate code from UML

```
MicroUMLCodeGenerator new
    diagram: d;
    generateCodeInPackage: p
```

Generate UML from package

```
MicroUMLClassDiagramGenerator
    classDiagramFromPackage: p
```

Limitations

Embedding a DSL in the host language simplifies implementation but also introduces trade-offs

1. Limited syntax

- Restricted by the host language
- Finite set of available symbolic selectors
- Potential conflict with other DSLs

2. Stateful interpretation

- Chained messages require internal mutable context
- Same selector may have different meaning depending on the context

3. Runtime validation

- Errors (wrong selectors) are detected only at runtime

Future Work

MicroUML is both a practical tool and a proof of concept

Improving MicroUML

- Support more UML diagrams
- Better IDE integration
- Improved synchronization with Pharo code
- Compare to Mermaid

Exploring other DSLs

- State machine DSL
- Cormas DSL for agent-based modelling
- Propose reusable infrastructure for parserless DSLs

UML for Cormas

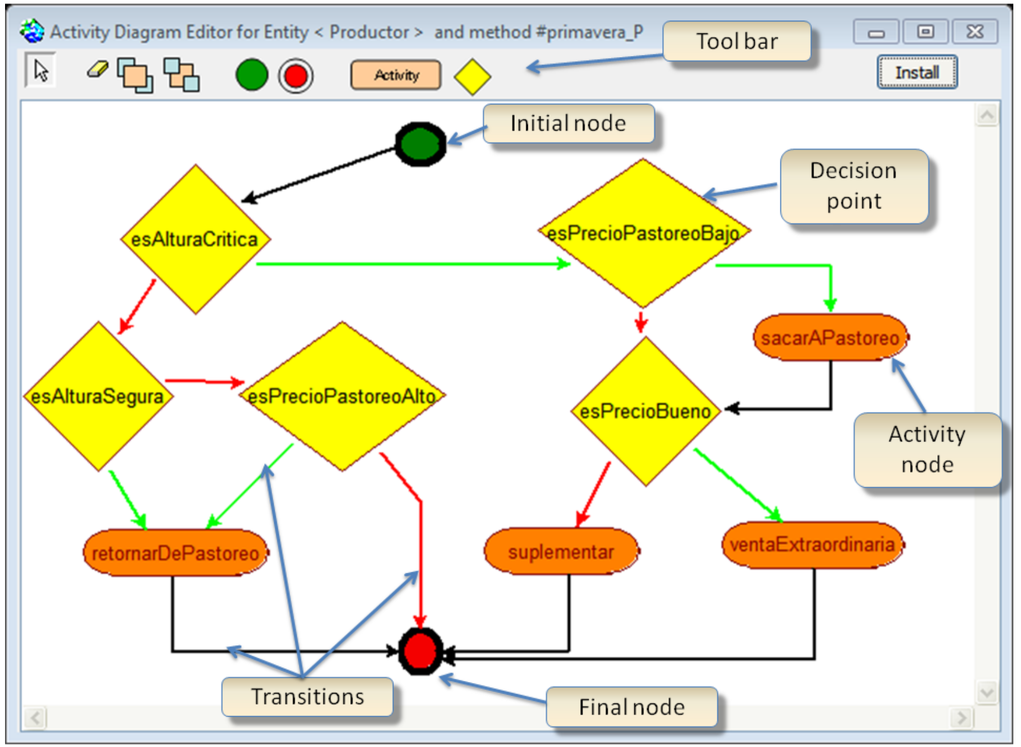
MicroUML revives and extends a long-term vision for participatory modelling

2014 – Executable UML for participatory modelling

- Executable UML activity diagrams
- Helps stakeholders to understand and modify models

Our next step

- Integrate MicroUML into Cormas
- Support live co-design of both structure and behavior



Bommel P., Dieguez F., Bartaburu D., Duarte E., Montes E., Pereira M., Corral J., Lucena C. and Morales H., (2014).
A Further Step Towards Participatory Modelling. Fostering Stakeholder Involvement in Designing Models by Using Executable UML. Journal of Artificial Societies and Social Simulation 17 (1) 6.

Contributions

Contribution 1:

We **designed MicroUML**, a practical textual UML DSL integrated with the Pharo environment.

Contribution 2:

We demonstrate that a non-trivial modelling DSL can be built **without introducing a grammar, parser or compiler** by leveraging Pharo's existing language and runtime infrastructure.

Try it yourself !

MicroUML on GitHub

https://github.com/olekscode/uUML