

# Modular oose

A platform for in-the-large & in-the-small  
reverse engineering

Nicolas Anquetil

# Agenda

Reverse engineering in the large/small

Moose, some background

Composable Meta-Model

Integrated Reverse Engineering Environment

Conclusion

# Agenda

## **Reverse engineering in the large/small**

Moose, some background

Composable Meta-Model

Integrated Reverse Engineering Environment

Conclusion

# Software maintenance



“Today, you will use 13 COBOL applications”  
[P. Nieuwbourg, 2012]

To remain useful, systems must evolve

- Adaptation to new needs (GUI, cloud, ...)
- Prepare for future evolution



# Software maintenance

Windows NT 3.1 (1993)

- $\approx 4$  a 5 MLOC

Windows server 2003

- $\approx 50$  MLOC

Windows 10

- ???

4  
2  
m



# Software maintenance

Ever living systems

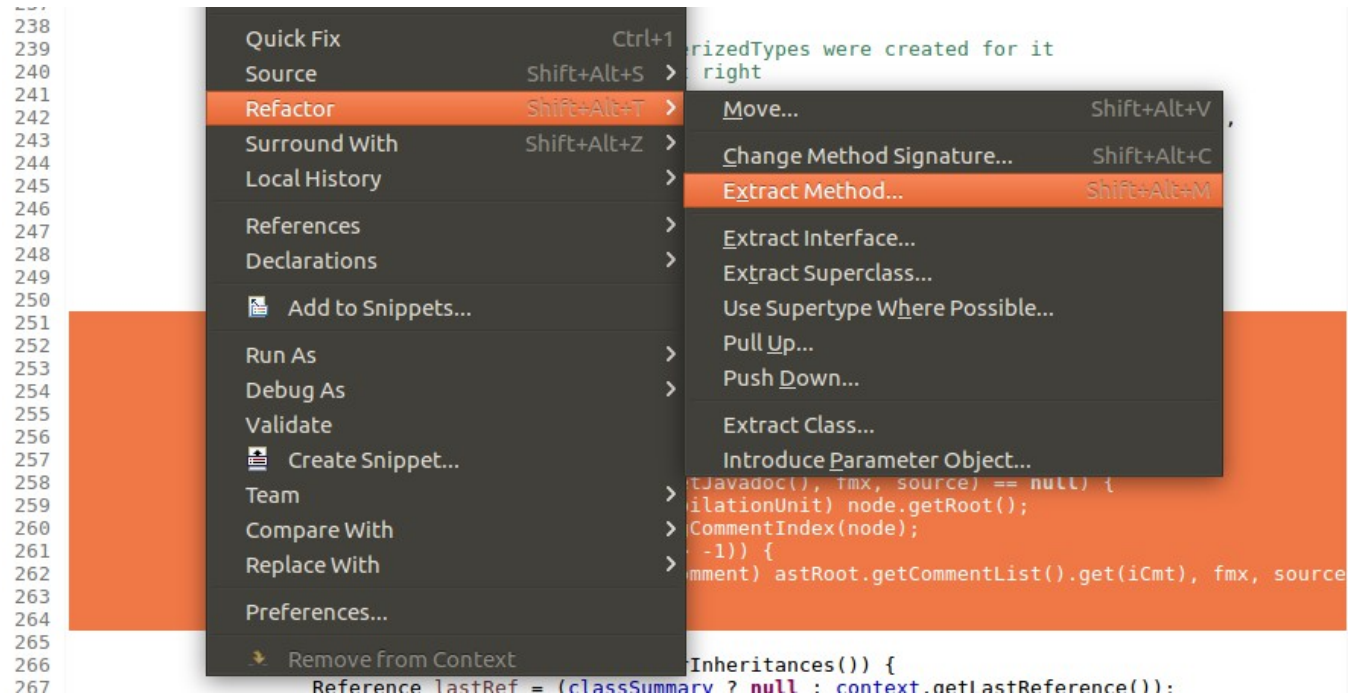
Intrinsic to software

Involves understanding a mathematical model (program)

Tools are needed

# Reverse engineering in the large/small

## Reverse engineering in the small



# Reverse engineering in the large/small

## Reverse engineering in the small

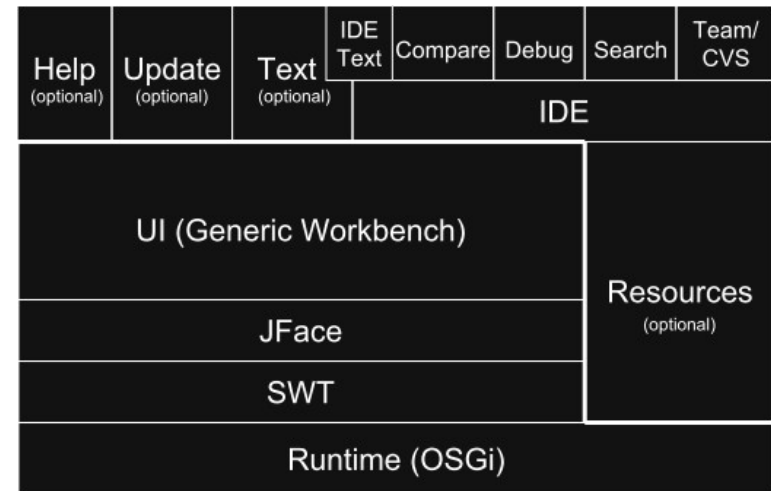
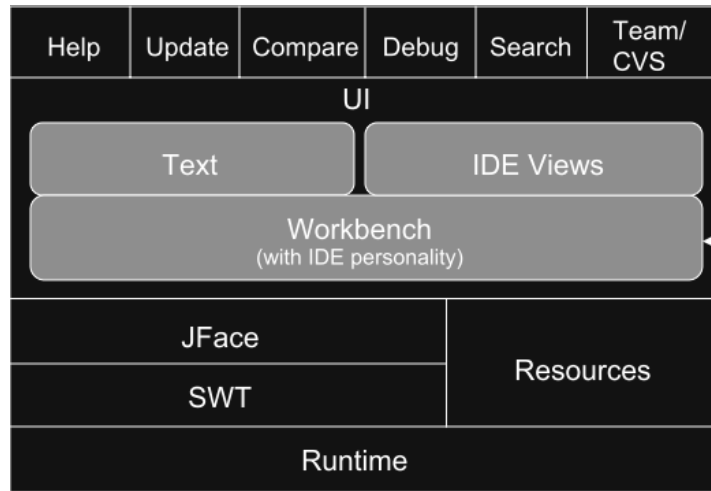
- On few entities ( $<10$ )
- Relevant code fits on 1 sheet of paper
- Complete understanding of each entity



# Reverse engineering in the large/small

## Reverse engineering in the large

- Eclipse v2.1 (Extensible IDE) v3.0 (Rich Client Platform)



# Reverse engineering in the large/small

## Reverse engineering in the large

- Restructure architecture
- Break a big class
- Introduce a design pattern (e.g. MVC, Hybernate)
- Migrate to a new library version
- ...

# Reverse engineering in the large/small

## Reverse engineering in the large

- Can/Should occurs regularly ( $\neq$  often) in the life of a system
- On many entities (tens, hundreds) / all system

# Reverse engineering in the large/small

## Reverse engineering in the small

- Detailed understanding
- Instructions, AST

## Reverse engineering in the large

- High level view on the system (but details matter)
- Packages, classes, dependencies

# Reverse engineering in the large/small

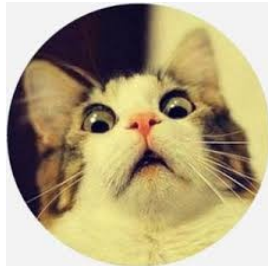
## Reverse engineering in the small

- Detailed understanding
- Instructions, AST

## Reverse engineering in the large

- High level view on the system (but details matter)
- Packages, classes, dependencies

+ generic tools



\* Frightened kitten

# Agenda

Reverse engineering in the large/small

**Moose, some background**

Composable Meta-Model

Integrated Reverse Engineering Environment

Conclusion

# oose, some background

A platform for software analysis

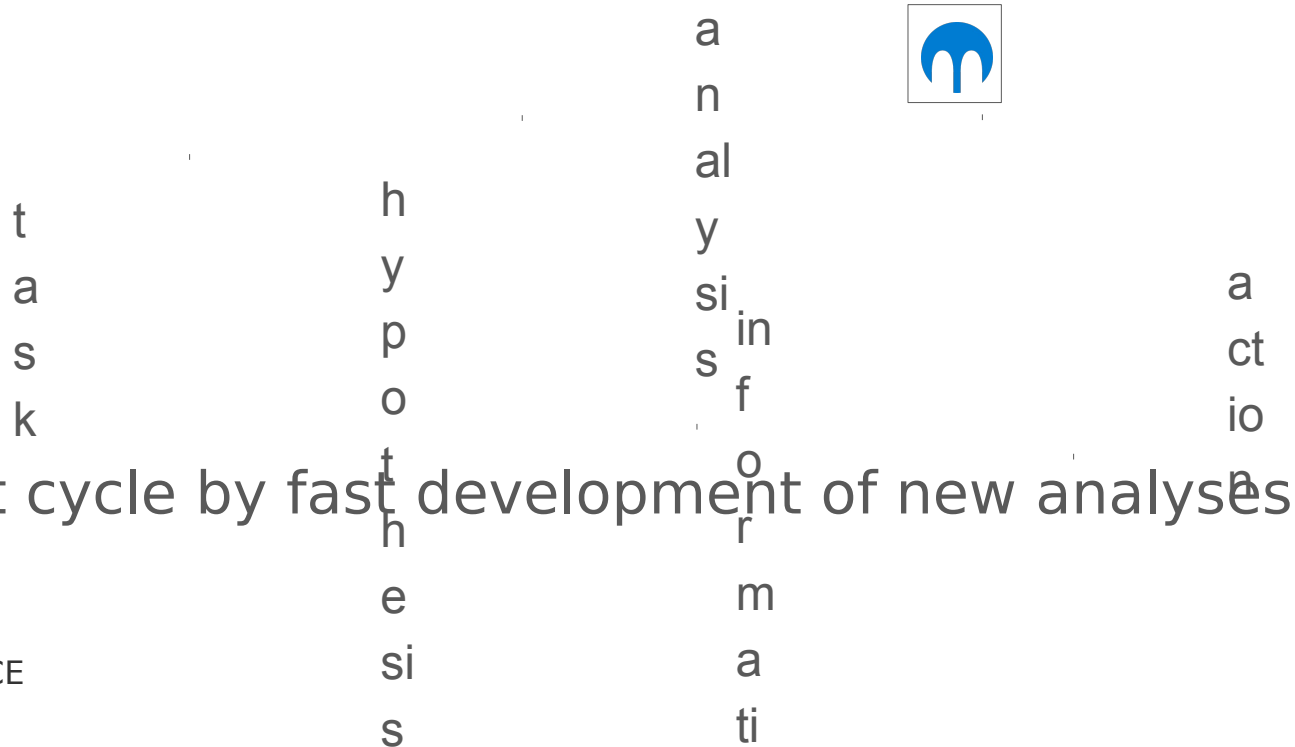
Based on a generic meta-model (Famix)

Developed since 1996

- <https://github.com/moosetechnology/moose-wiki>
- <https://github.com/moosetechnology>

# oose, some background

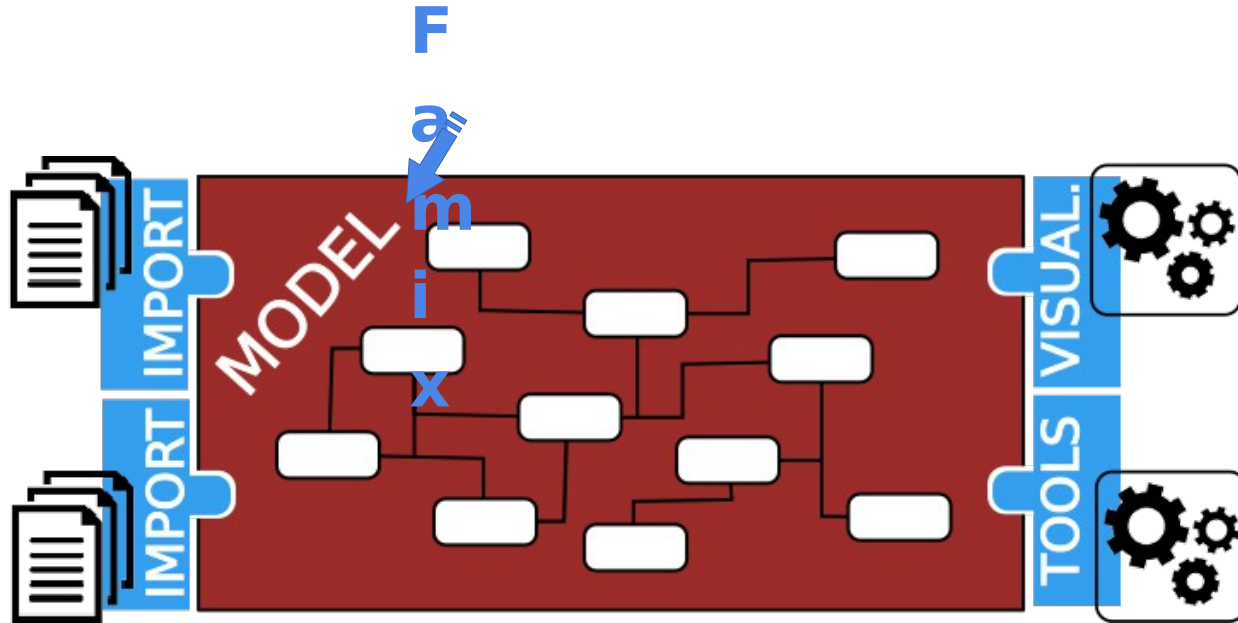
A platform for software analysis





# moose, some background

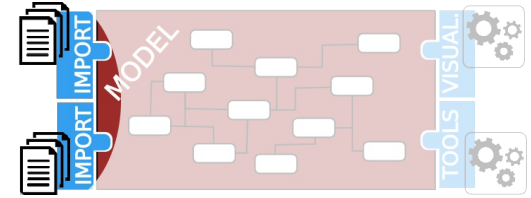
A platform for software analysis



# moose, some background

## Importers

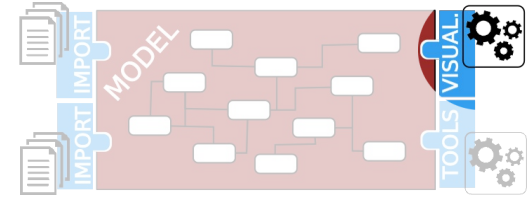
- Populate a model from existing documents
- Source code (4D, Ada, C/C++, COBOL, Fortran, Java, Mantis, Pharo, PowerBuilder, SQL, TypeScript, VBA, ...)
- HTML, XML, CSV, ...
- Bugs, authors, commits



# moose, some background

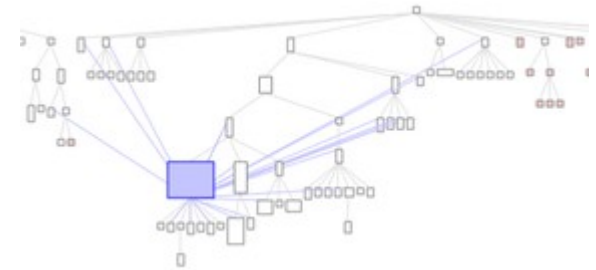
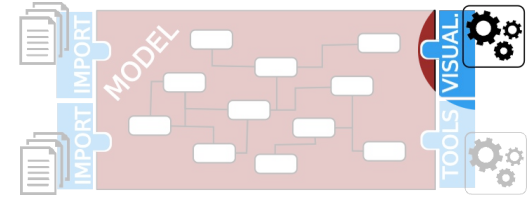
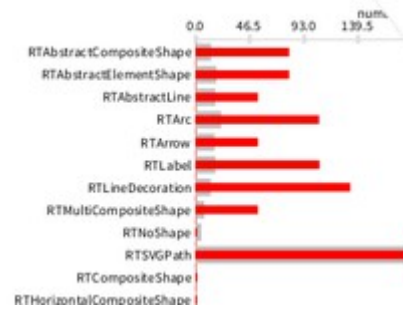
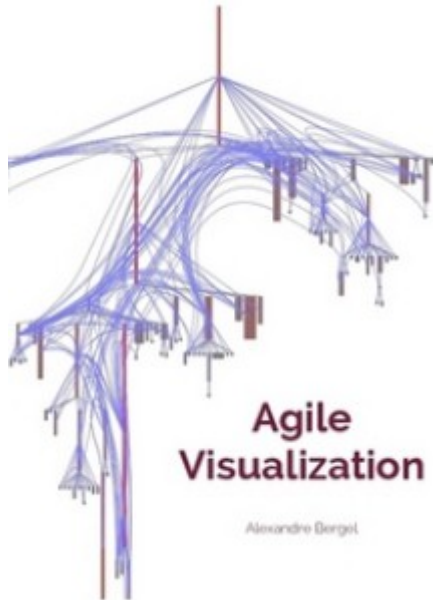
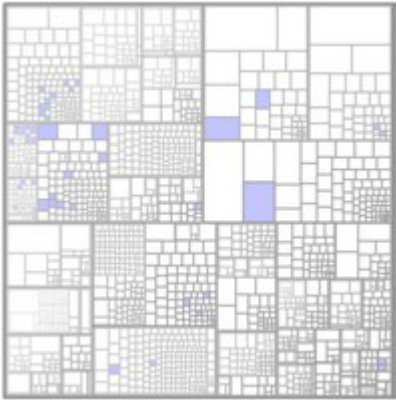
## Visualizations

- Roassal: DSL to script visualizations
  - “Agile Visualization” : explore data through visualizations
  - <http://agilevisualization.com/>
- Telescope: Tool with configurable predefined visualizations
  - Rendering with Roassal
  - Rendering with Cytoscape (JavaScript)



# moose, some background

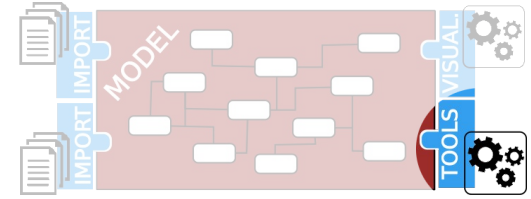
## Visualizations



# moose, some background

## Tools

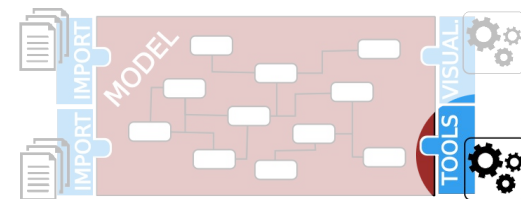
- MooseQuery: DSL to query a model
- Software engineering metrics (cohesion/coupling, Chidambler & Kemerer metrics, LCOM, cyclomatic complexity, ...)
- Tagging (labels and/or virtual entities)
- Data Mining algorithms



# oose, some background

## Moose Query

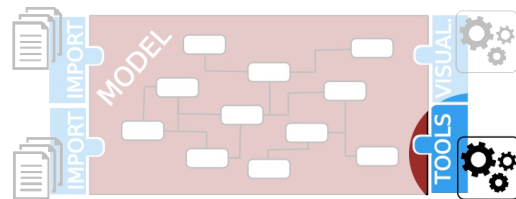
- <https://moosequery.ferlicot.fr/>
- API to programmatically query FamixNG models
  - Entities respecting some condition (name, metrics, ...)
  - Containment navigation (parent, children)
  - Dependency navigation (clients/providers, invoked methods,...)



# oose, some background

## Moose Query

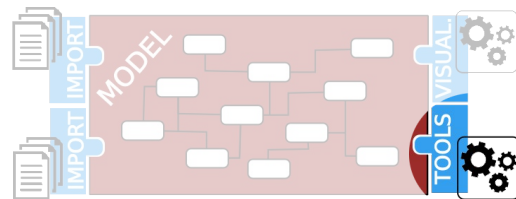
- Java analysis: “All provider packages for package *p* via method calls?”  
(p `queryOutgoing:` FamixInvocation) `atScope:` FamixPackage
- SQL analysis: “All stored procedures accessing a given table column *c*?”  
c `queryIncomingDependencies` `atScope:` FamixSQLStoredProcedure



# moose, some background

## Moose Query

- Java analysis: “All provider packages for package  $p$  via method calls?”  
(p `queryOutgoing:` FamixInvocation) `atScope:` FamixPackage
- SQL analysis: “All stored procedures accessing a given table column  $c$ ?”  
c `queryIncomingDependencies` `atScope:`  
FamixSQLStoredProcedure

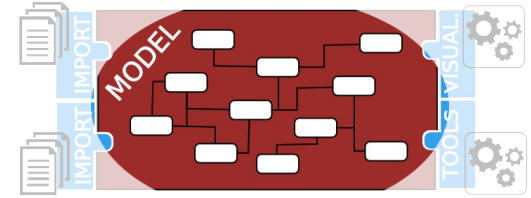
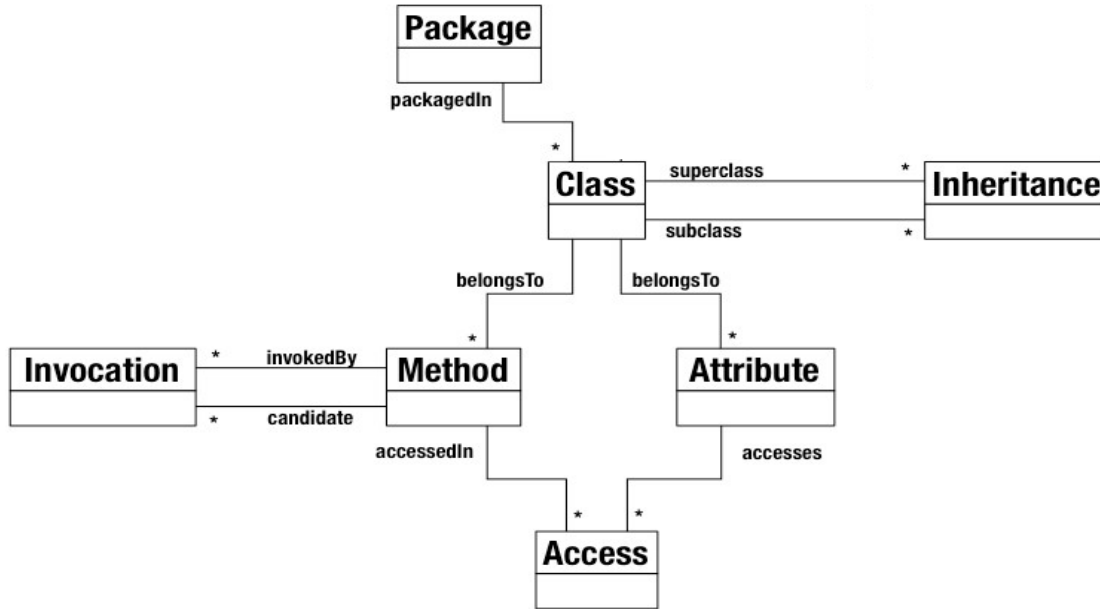


**F**  
**O**  
**R**



# moose, some background

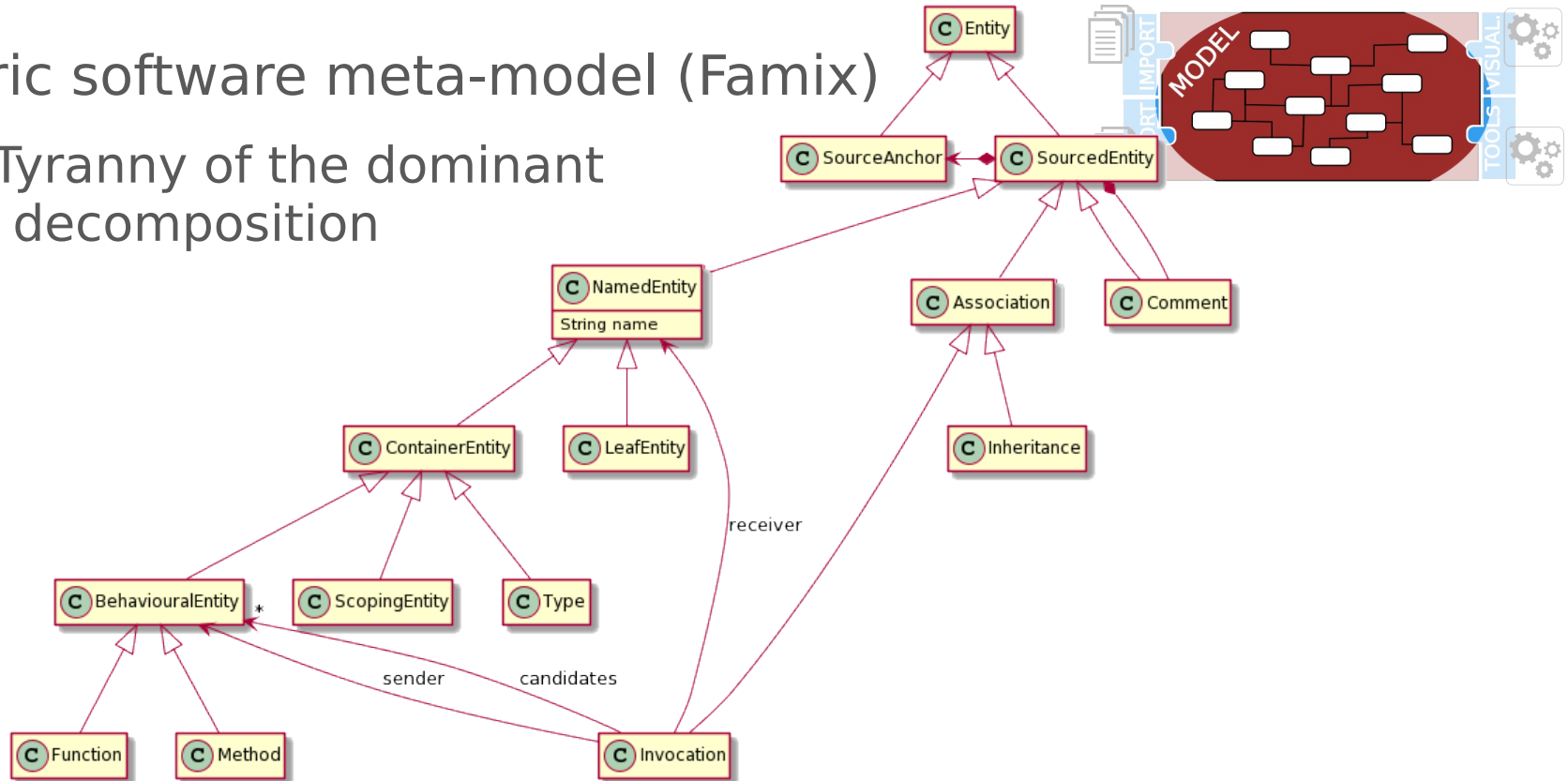
Generic software meta-model (Famix)



# moose, some background

## Generic software meta-model (Famix)

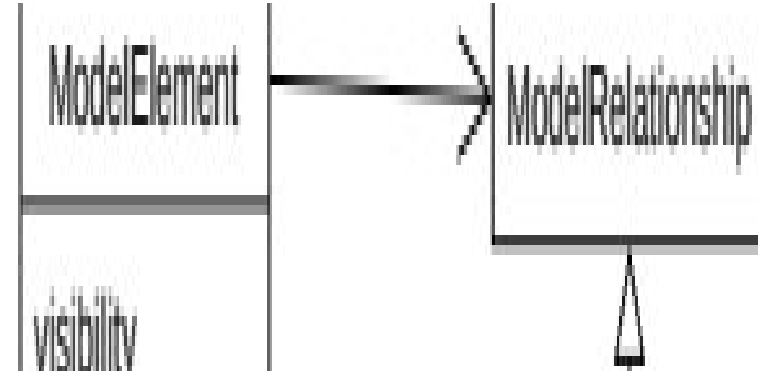
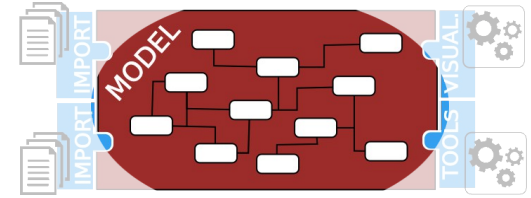
- Tyranny of the dominant decomposition



# Software Meta-Modeling

## Generic software meta-model

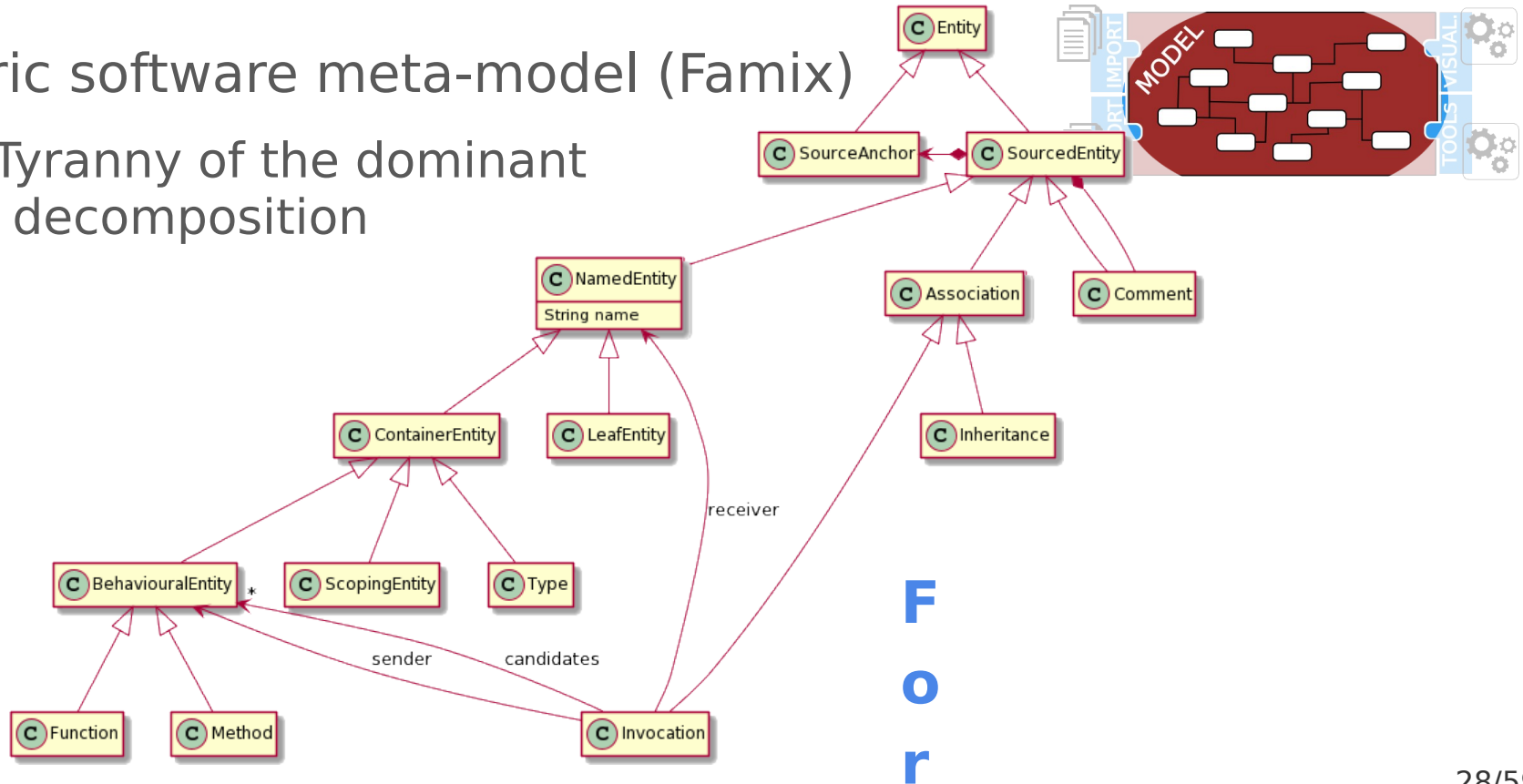
- Dagstuhl Middle Meta-model
- Tyranny of the dominant decomposition



# moose, some background

## Generic software meta-model (Famix)

- Tyranny of the dominant decomposition



# Agenda

Reverse engineering in the large/small

Moose, some background

## **Composable Meta-Model**

Integrated Reverse Engineering Environment

Conclusion

# Composable Meta-Model

~~Famix: One generic meta-model for all languages~~

Create specialized meta-model for each language

FamixNG: Bare bones entities + composable properties

- Classes have *attributes* and *methods* and *inheritance*
- + *visibility* (public, private, protected, friend, ...)
- + *partial* classes + *extension* methods + use *traits*

# Composable Meta-Model

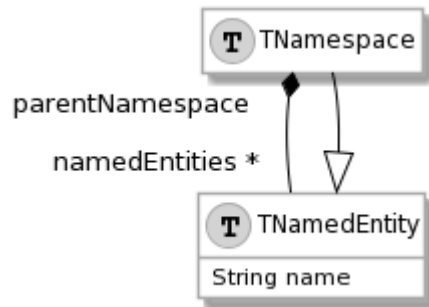
## FamixNG

- $\simeq$  100 traits (i.e. a set of methods that classes can *use* with a kind of “multiple inheritance”)

# Composable Meta-Model

## FamixNG

- $\approx 100$  traits (i.e. a set of methods that classes can use with a kind of “multiple inheritance”)
- Ex: `TNamedEntity` *used* by entities that have a name
- Ex: `TNamespace` *used* by entities that contain `TNamedEntity`

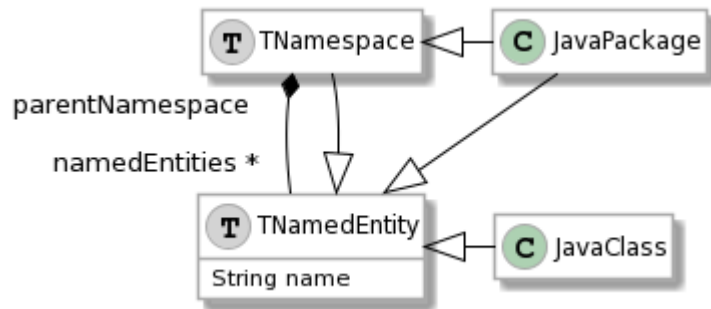




# Composable Meta-Model

## FamixNG

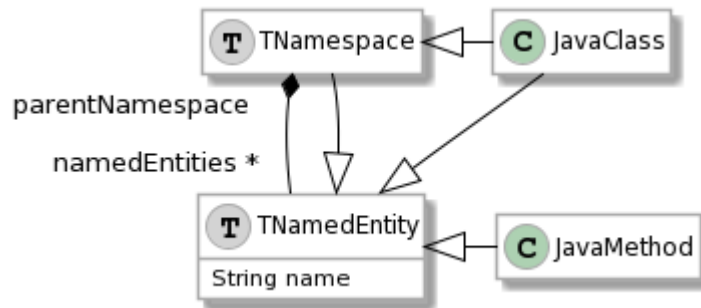
- $\approx 100$  traits (i.e. a set of methods that classes can use with a kind of “multiple inheritance”)
- Ex: `TNamedEntity` *used* by entities that have a name
- Ex: `TNamespace` *used* by entities that contain `TNamedEntity`



# Composable Meta-Model

## FamixNG

- $\approx$  100 traits (i.e. a set of methods that classes can use with a kind of “multiple inheritance”)
- Ex: `TNamedEntity` *used* by entities that have a name
- Ex: `TNamespace` *used* by entities that contain `TNamedEntity`



# Composable Meta-Model

Generic TClass in FamixNG *uses*:

- TType (which *uses* TNamedEntity)
- TWithComment
- TPackageable
- TWithInheritances
- TWithAttributes
- TWithMethods
- TInvocationsReceiver

# Composable Meta-Model

JavaClass *uses*:

- TClass
- TWithVisibility
- TWithExceptions

PharoClass *uses*:

- TClass
- TUsesTraits
- TWithExtentions

# Composable Meta-Model

TMethod uses:

- TNamedEntity
- THasSignature
- TTypedEntity
- TWithClassScope
- TInvocable
- TWithImplicitVariables
- TWithLocalVariables
- TWithParameters
- TWithReferences
- TWithStatements

# Meta-Model builder

“Builders” to create metamodels

- DSL (inspired by PlantUML)
- Can import other meta-models to compose meta-models together (ex: Java program with SQL queries)

# Composable Meta-Model

## FamixNG

- Association traits (7):  
Inheritance, Access, Invocation, Reference, UseTrait, IncludeFile
- Technical traits (12):  
SourceAnchor, metrics, queries
- Property traits(46):  
name, comments, Typed, visibility, Invocable, ...
- Terminal/Core/Entity traits (38):  
Class, Method, Attribute, Parameter, Exception, Function, ...

# Meta-Model builder

New language meta-models built from FamixNG traits library  
“Builders” to create meta-models



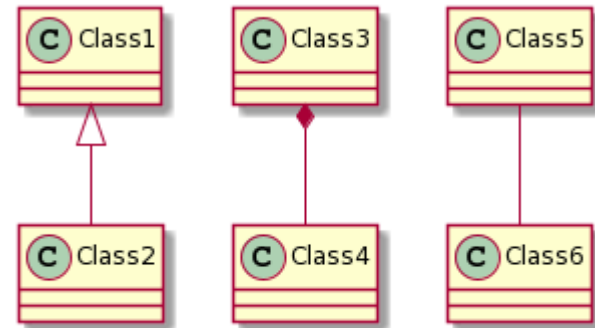
# Meta-Model builder

New language meta-models built from FamixNG traits library

“Builders” to create meta-models

- DSL (inspired by [PlantUML.com](http://PlantUML.com))

```
@startuml
Class1 <|-- Class2
Class3 *-- Class4
Class5 -- Class6
@enduml
```



# Meta-Model builder

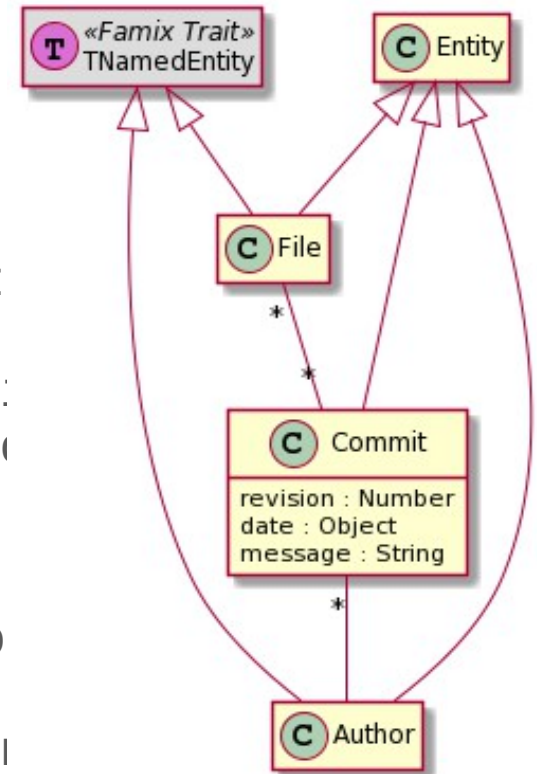
## Example: Meta-model for commits

### Define classes

- entity := builder newClassNamed: 'Entity'
- file := builder newClassNamed: 'File'.
- commit := builder newClassNamed: 'Commit'
- author := builder newClassNamed: 'Author'

### Define properties

- commit property: 'revision' type: #Number
- commit property: 'date' type: #Object.
- commit property: 'message' type: #String



# Meta-Model builder

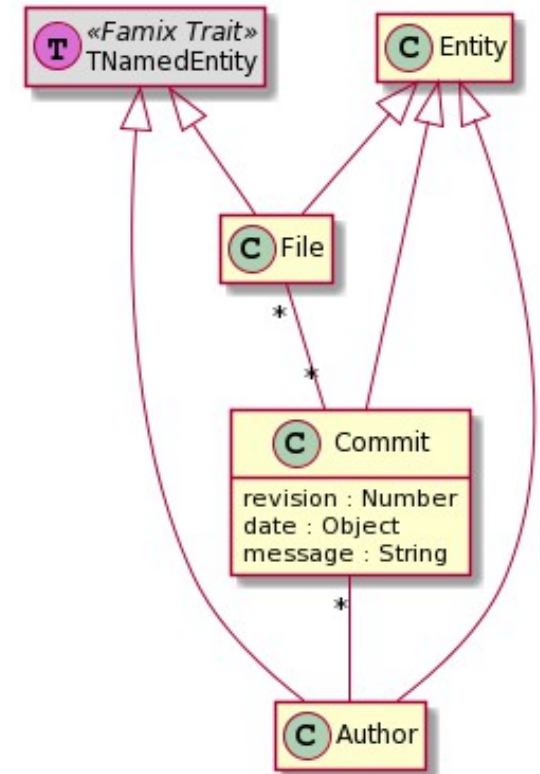
Example: Meta-model for commits

Define inheritances

- file --|> entity.  
file --|> TNamedEntity.  
commit --|> entity.  
author --|> entity.  
author --|> TNamedEntity.

Define relations

- file \*- \* commit.  
commit \*- author.



# Agenda

Reverse engineering in the large/small

Moose, some background

Composable Meta-Model

**Integrated Reverse Engineering Environment**

Conclusion

# Integrated Reverse Engineering Environment

Reverse engineering involves many tasks

- Visualization, query, metrics, navigation, dependency analysis, control flow/data flow analysis
- 

This calls for many specialized tools collaborating

The tools must be generic (meta-model agnostic)

The tools must collaborate

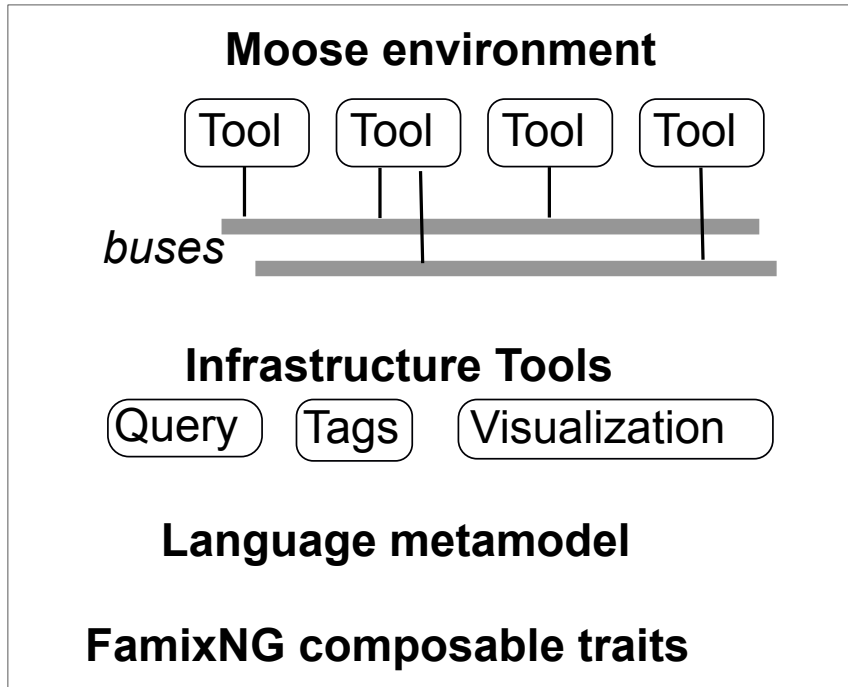
# ModMoose

## ModMoose an Integrated Reverse Engineering Environment IREE

- Iree is of Hebrew origin and means: Gift of God
- In Jamaican English, iree (or irie) means: nice, good, or pleasing; Iree is about being calm pleasant and taking life a day at a time
-

# ModMoose

## ModMoose an Integrated Reverse Engineering Environment



# ModMoose

## ModMoose an Integrated Reverse Engineering Environment

- The environment centralizes data and tool interactions
- Tools are focusing on a single task: *e.g.*, the Query Browser works on a set of model entities and produces another set of entities
- Tools communicate through buses, they “read” model entities on their bus(es) and “write” entities back on their bus(es)



# Tools

*Model Browser* (imports/selects models)

*Entity Inspector* (Properties of selected entities + values)

*Query Browser* (GUI for MooseQuery)

*Dependency Graph Browser* (graph w/ incoming/outgoing dependencies of entities)

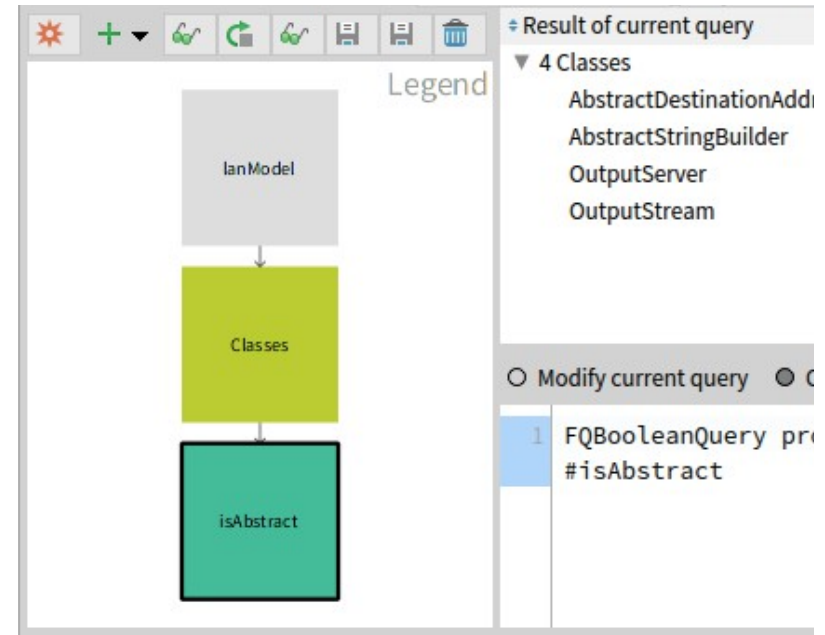
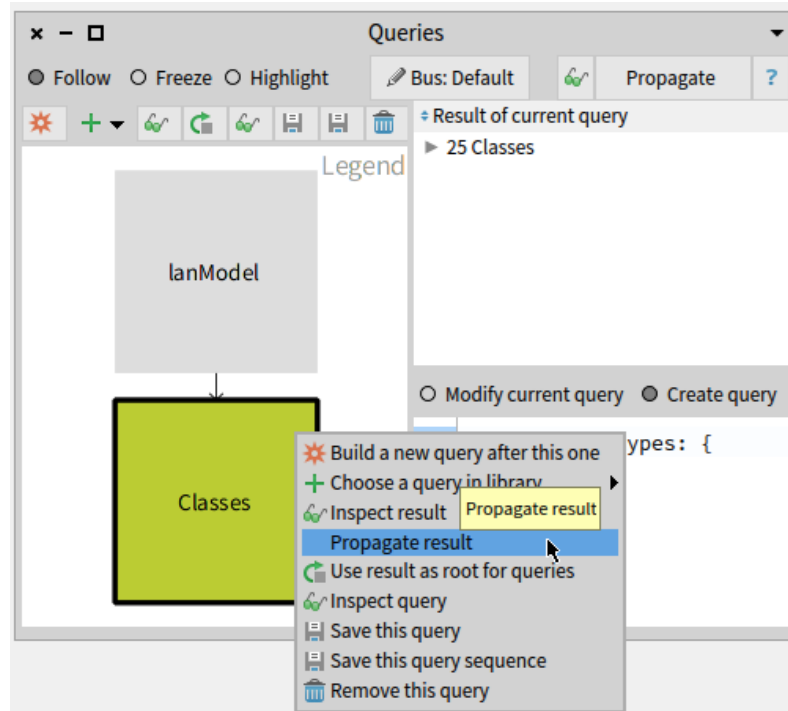
*Duplication Browser* (shows code clones occurrences)

*Source Code* (Listing of source code)

*Logger* (Records each step with entities that pass on a bus)

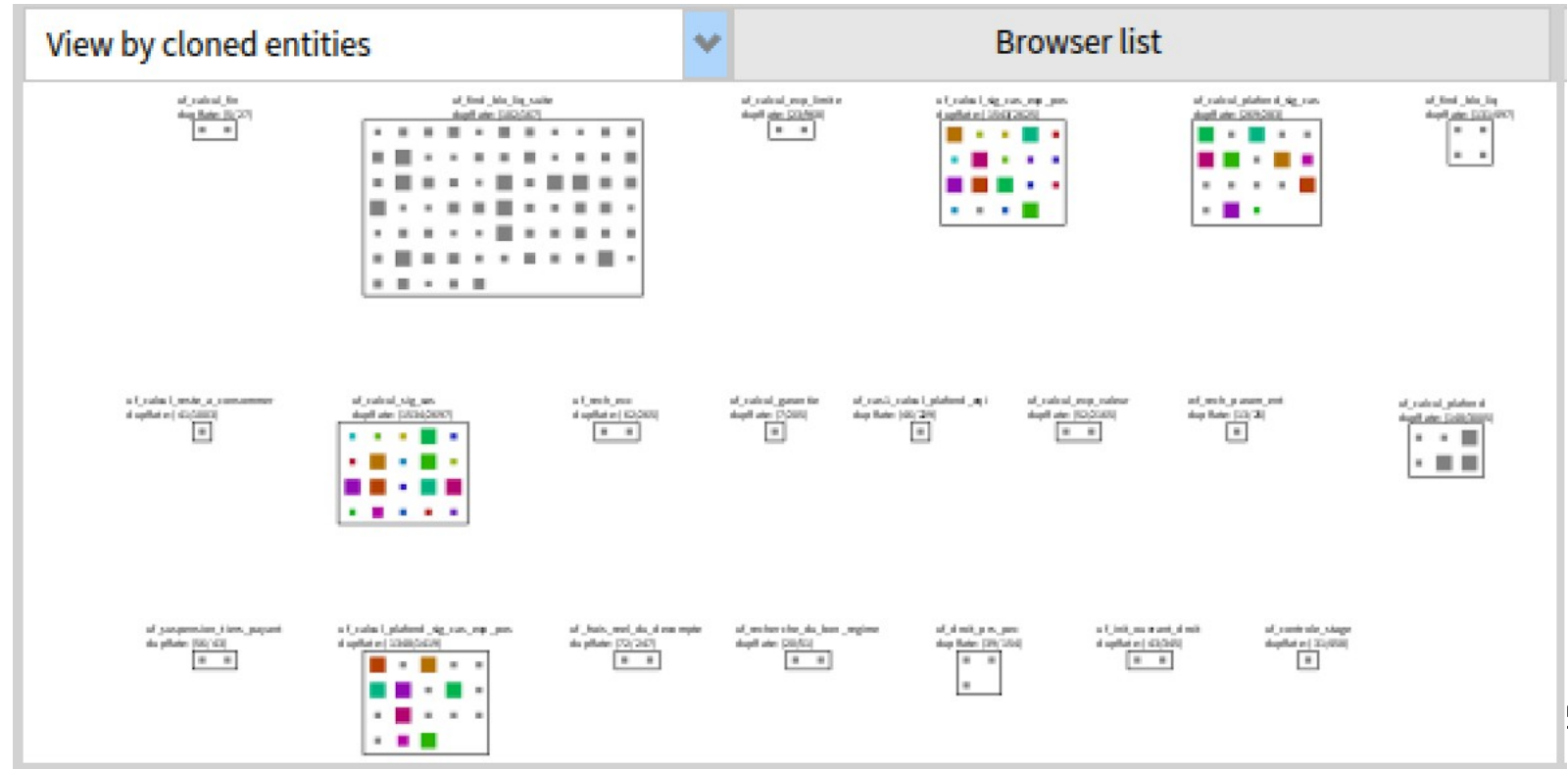
# Tools

## Query Browser



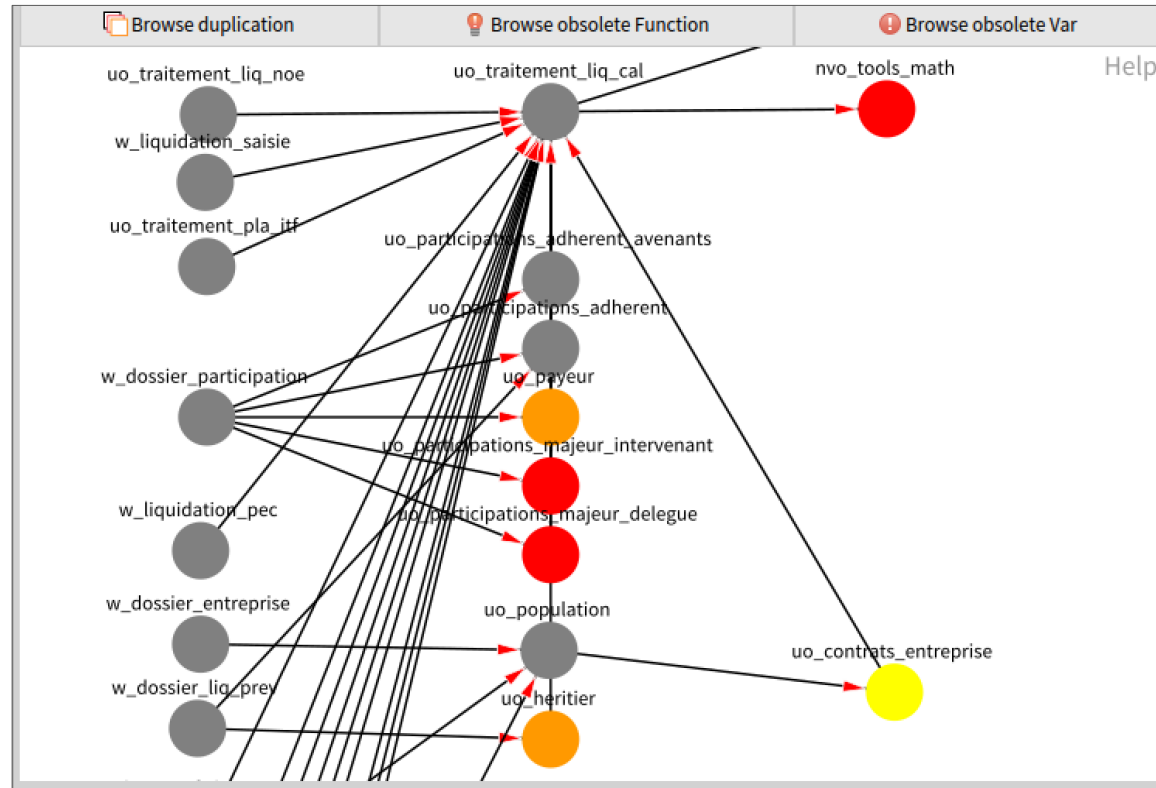
# Tools

## Duplication Browser



# Tools

## Dependency Graph Browser



# Tools

## Entity Inspector

Navigation

#'Smalltalk::AbstractFont'

#'Smalltalk::AnnouncementsAPIDocumentation'

#'Smalltalk::AnnouncementsHelp'

#'Smalltalk::AnnouncementsTutorial'

#'Smalltalk::Array'

#'Smalltalk::ArrayedCollection'

#'Smalltalk::AthensAbstractPaint'

#'Smalltalk::AthensAbstractShape'

#'Smalltalk::AthensAffineTransform'

#'Smalltalk::AthensBalloonEngine'

#'Smalltalk::AthensBalloonGradientPaint'

#'Smalltalk::AthensBalloonImagePaint'

#'Smalltalk::AthensBalloonLinearGradient'

#'Smalltalk::AthensBalloonPaint'

#'Smalltalk::AthensBalloonPath'

#'Smalltalk::AthensBalloonRadialGradient'

#'Smalltalk::AthensBalloonSolidColorPaint'

#'Smalltalk::AthensBalloonStrokePaint'

#'Smalltalk::AthensBalloonSurface'

#'Smalltalk::AthensCairoCanvas'

#'Smalltalk::AthensCairoPath'

Navigation

Tree

Moose Properties

Fame

Name

Type

Opposite

Derived?

Container?

IsTarget?

IsSource?

annotationInstantiation

TAnnotationInstantiation

annotatedEntity

true

false

false

false

attributes

TAttribute

parentType

true

false

false

false

comments

TComment

container

true

false

false

false

declaredSourceLanguage

TSourceLanguage

sourcedEntities

false

false

false

false

definedAnnotationTypes

TAnnotationType

annotationTypes

true

false

false

false

exceptions

TException

exceptionClass

true

false

false

false

extendedMethod

Method

true

false

false

false

incomingReference

TReference

target

true

false

false

false

methods

TMethod

parentType

true

false

false

false

parentPackage

TPackage

childEntities

false

true

false

false

receivingInvocation

TInvocation

receiver

true

false

false

false

sourceAnchor

TSourceAnchor

element

true

false

false

false

subInheritance

TInheritance

superclass

true

false

false

false

superInheritance

TInheritance

subclass

true

false

false

false

typeContainer

TWithTypes

types

false

true

false

false

typedEntities

TTypedEntity

declaredType

true

false

false

false

types

TType

typeContainer

true

false

false

false

# Communication Buses

Tools “read” and “write” model entities on buses

Each tool can be attached to 0 to n buses

Can have several instances of the same tool concurrently

Ex: Compare dependencies of two set of entities

- Bus1: QueryBrowser1 + DependencyGraphBrowser1
- Bus2: QueryBrowser2 + DependencyGraphBrowser2

# Communication Buses

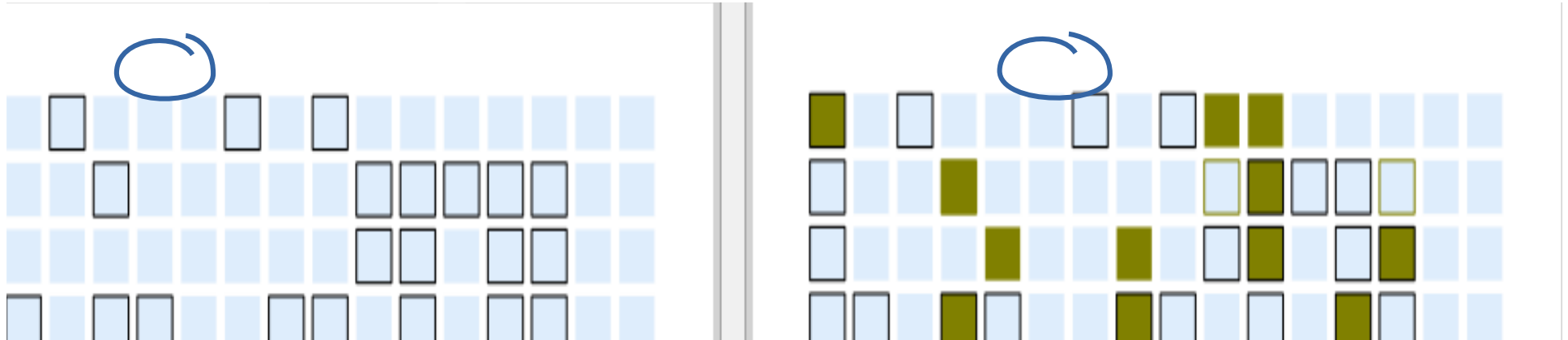
Logger tool can be set as a bridge between all buses

- Listens to all buses (records all steps in session)
- Forwards its activity on all buses (to synchronize them)

# Tools

## Fine control of tool behavior

- Follow: Display incoming entities, produces outgoing entities
- Highlight: Highlight incoming entities in “frozen” display
- Frozen: Frozen display but, produces outgoing entities





# Agenda

Reverse engineering in the large/small

Moose, some background

Composable Meta-Model

Integrated Reverse Engineering Environment

**Conclusion**

# WrapUp

## FamixNG

- Composition of programming language meta-models from basic traits

## ModMoose

- Interactive Reverse Engineering Environment
- Specialized tools communicating through buses

# Modular oose

Nicolas Anquetil

`nicolas.anquetil@inria.fr`

