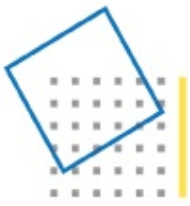
Report About:

MOOSE

FAMIX

VerveineJ

By: Ra'fat A. AL-msie'deen



Laboratoire
d'Informatique
de Robotique
et de Microélectronique
de Montpellier

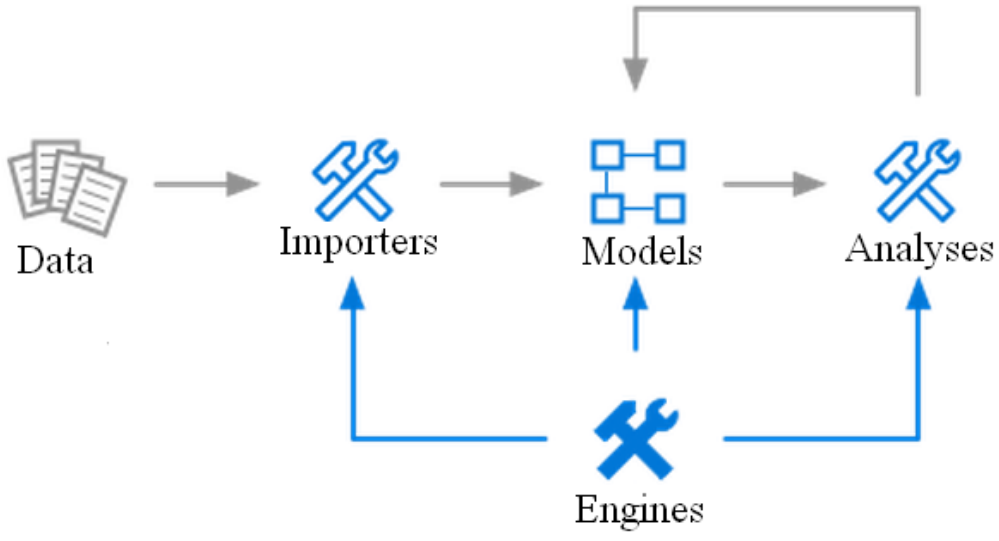
MOOSE:

Moose: is a platform for software and data analysis. It is an open source project since 1996. It is supported by several research groups around the world, and it is increasingly adopted in industrial projects [1].

Moose is an extensive platform for software and data analysis. It offers multiple services ranging from importing and parsing data, to modelling, to measuring, querying, mining, and to building interactive and visual analysis tools [1].

Moose is an open-source platform for expressing analyses of software systems and of data in general. Its main goal is to assist and enable a human in the process of understanding large amounts of data. It addresses several categories of people: [1] researchers in the area of software analysis, mining and reverse engineering, [2] engineers and architects who want to understand systems and data, and [3] tool builders.

Moose is a generic platform for engineers that want to understand data in general and software systems in particular. From a conceptual point of view, Moose is organized as follows.



[General workflow of working with Moose]

- The input is always some piece of data [software system written in Java or, a set of configuration files written in XML or, some meta-data about your system].
- This data is loaded in Moose via importers.
- When the data import, the data is stored in models. Based on these models you can start performing various kinds of analyses.
- Analysis such as: Tons of metrics, a multitude of queries, various data mining and graph-based algorithms, duplication detection, interactive visualizations of all sorts, or data browsers.
- Moose is more than a tool, Moose is a platform, and Moose helps you to build your own tools. Through Moose you can:
    - Build new importers for new data sets,
    - Define new models to store the data, and
    - Create new analysis algorithms and tools such as: complex graph visualizations, charts, new queries, or even complete browsers and reporting tools altogether.
- Dealing with models: The core of Moose offers support for storing and manipulating entities in models.

FAMIX:

FAMIX: is a family of meta-models. The Core of FAMIX is a language independent meta-model that describes the static structure of object-oriented software systems. The FAMIX Meta model supports multiple object-oriented languages [1].

MSE:

MSE is the generic file format used for import-export in Moose. Similar to XML, MSE is generic and can describe any model. MSE is built as part of the Fame project [1].

VerveineJ:

VerveineJ is an open-source MSE exporter built by Nicolas Anquetil. It is a Java 6 application based on JDT and it is available from the command line [1].

VerveineJ allows extracting information from the source code of Java systems and exporting it for the Moose platform. From this, Moose allows performing different analysis on how the system is structured, interactions between its components, quality, etc [1].

inFamix:

inFamix is a parser for Java and C/C++, inFamix parser is based on the scalable inFusion platform. The parser is available as a free command line tool and it exports MSE [1].

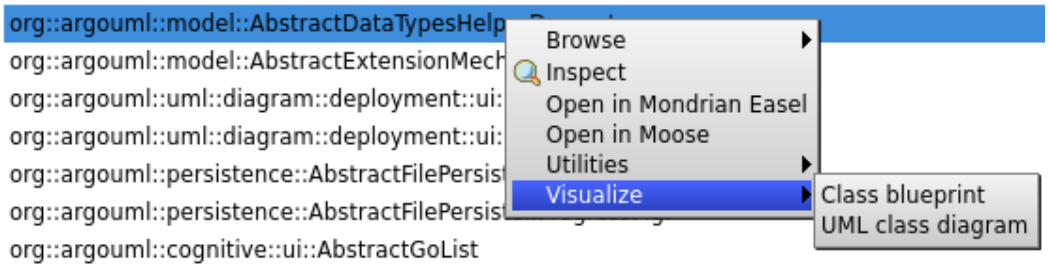
[Note]:

Plugins can define their own importers and make them appear in the menu of the Moose Panel.

[Everything is an Entity]:

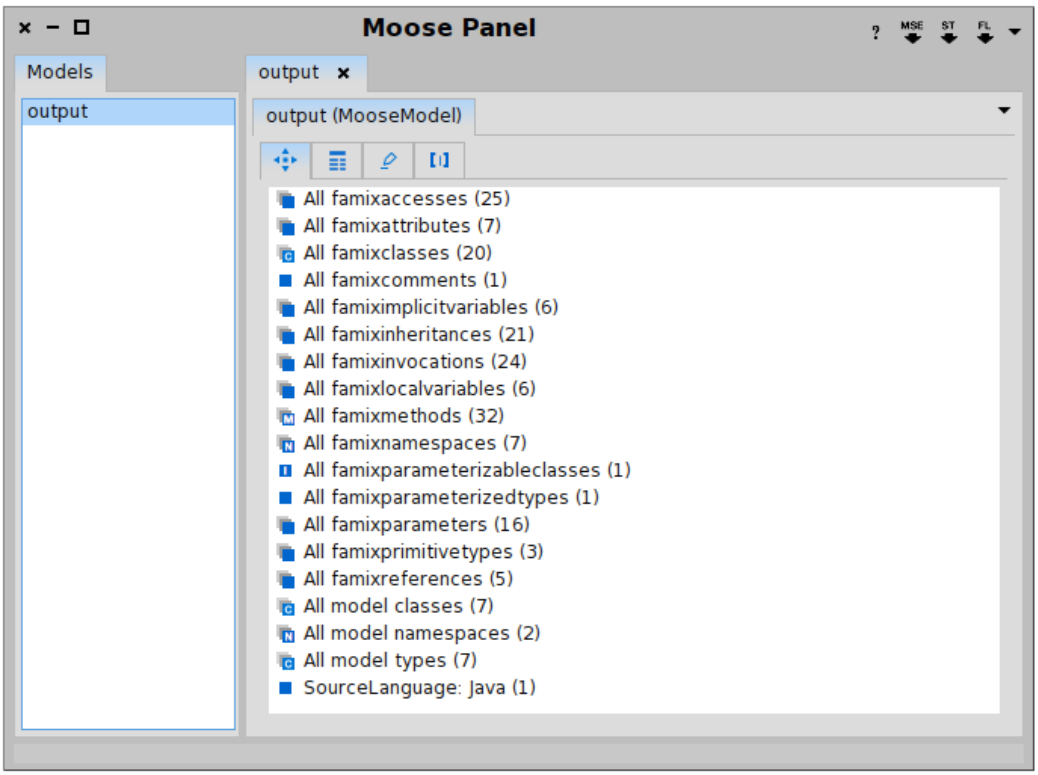
In object-orientation everything is an Object, in Moose everything is an Entity. More specifically, everything you see is an entity representing data. The simplest possible

interaction is by clicking on it and spawning a contextual menu with the possible services that can be called on this entity.



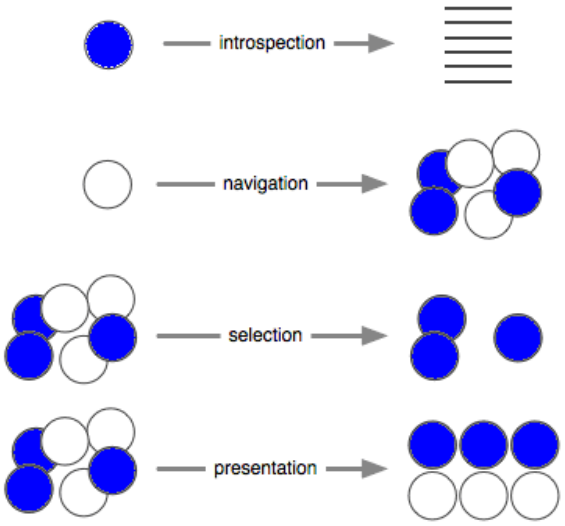
[Example Moose menu for a class]

Moose Finder: The basic user interface of Moose is the Moose Finder, and its goal is to offer a generic user interface for navigating large models. The Moose Finder is embedded in the Moose Panel and can be obtained by clicking on a model from the left pane.



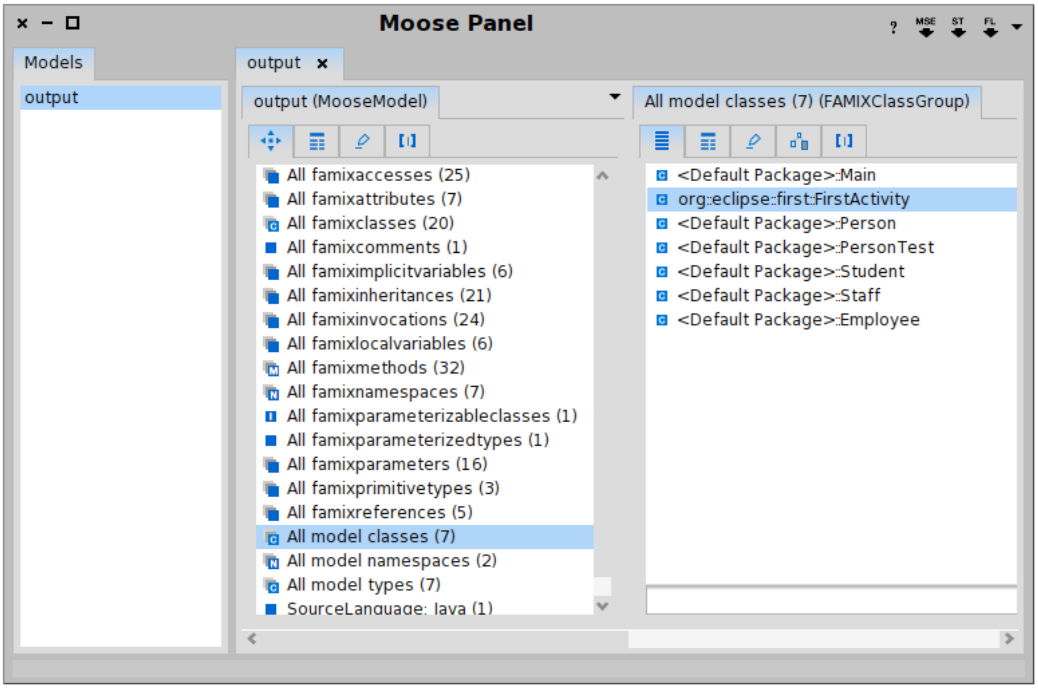
[The Moose Finder open on a model]

- The Moose Finder was designed to support four basic analysis actions [1]:
  - Introspection: given an entity, retrieve its properties;
  - Selection: given a group of entities, retrieve those that match a set of criteria;
  - Navigation: given an entity, retrieve other entities that are related;
  - Presentation: given a group of entities, arrange them according to a set of criteria.



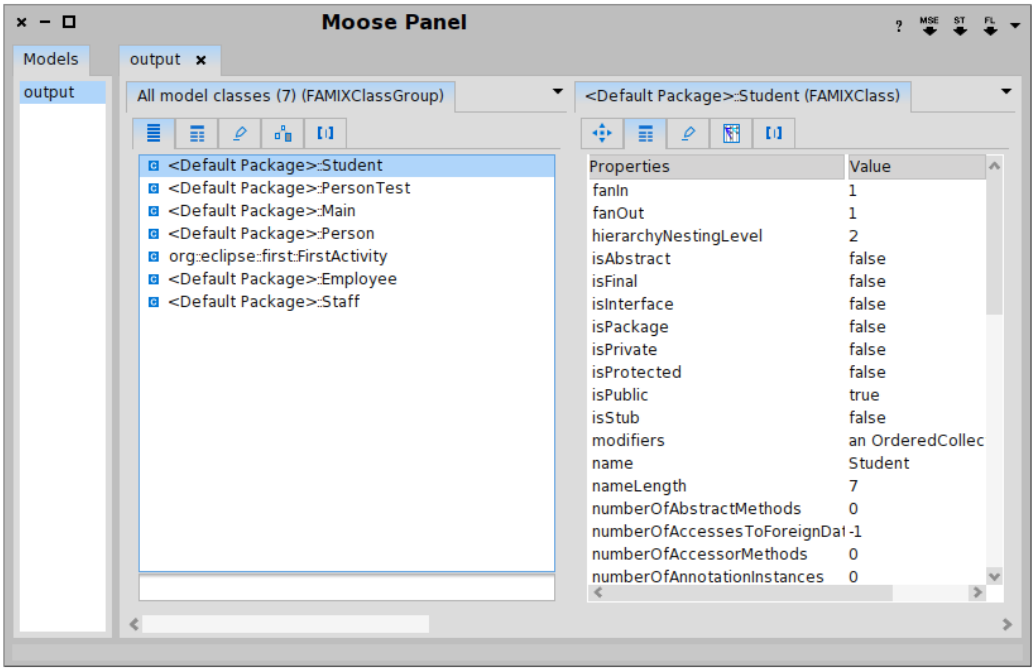
[Analysis Actions]

- Introspection is obtained by spawning another pane with the details of the currently selected entity.



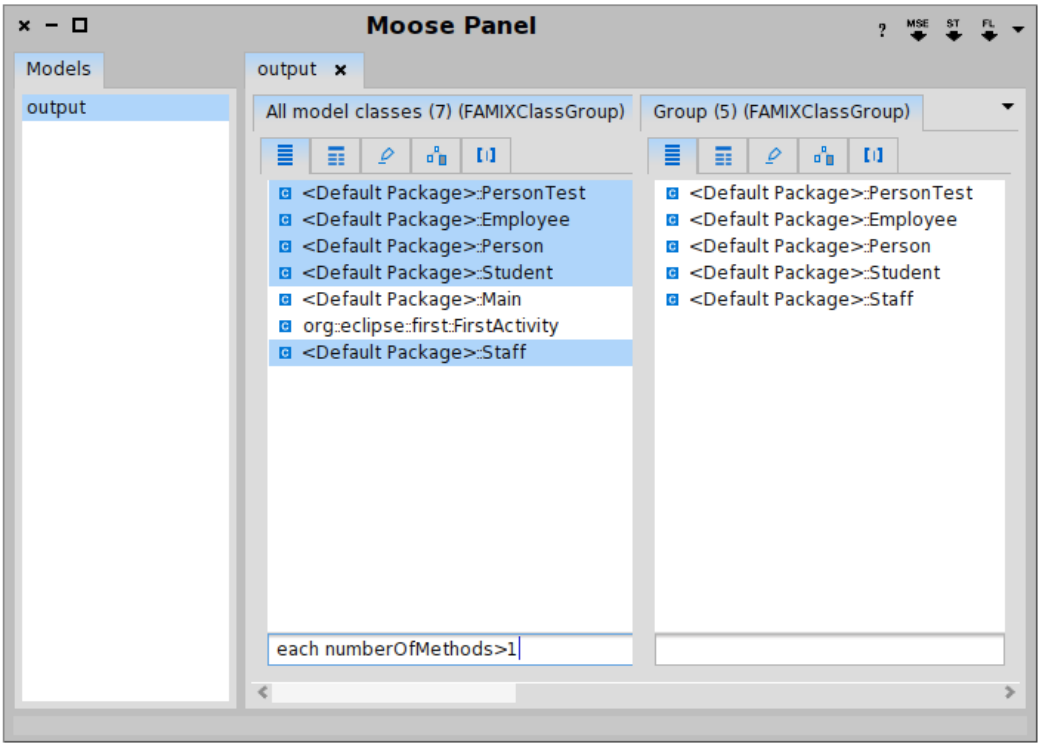
[Finder Introspecting a Group of Classes]

- Similarly, the picture below shows the detailed properties of one class.

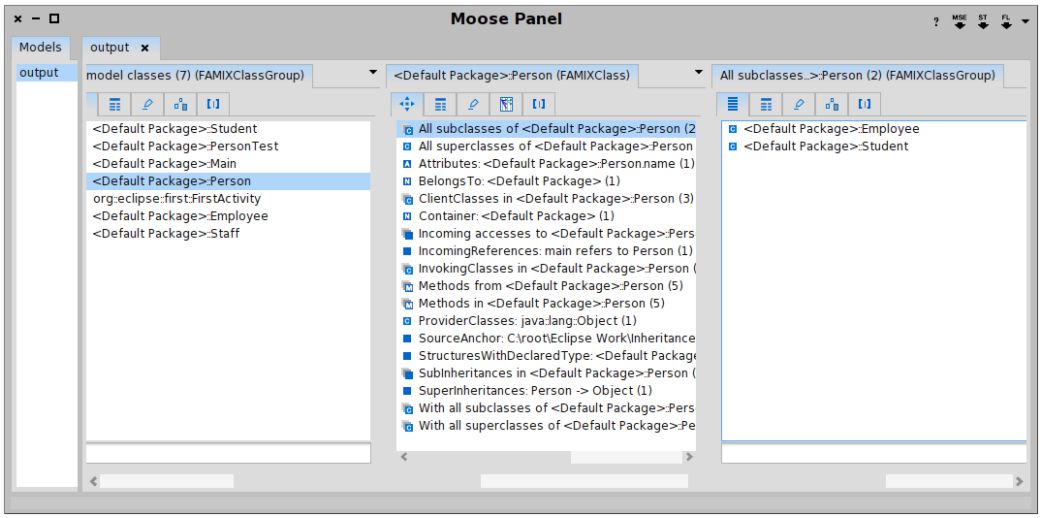


[Finder Introspecting the Properties on a Selected Class / Student Class]

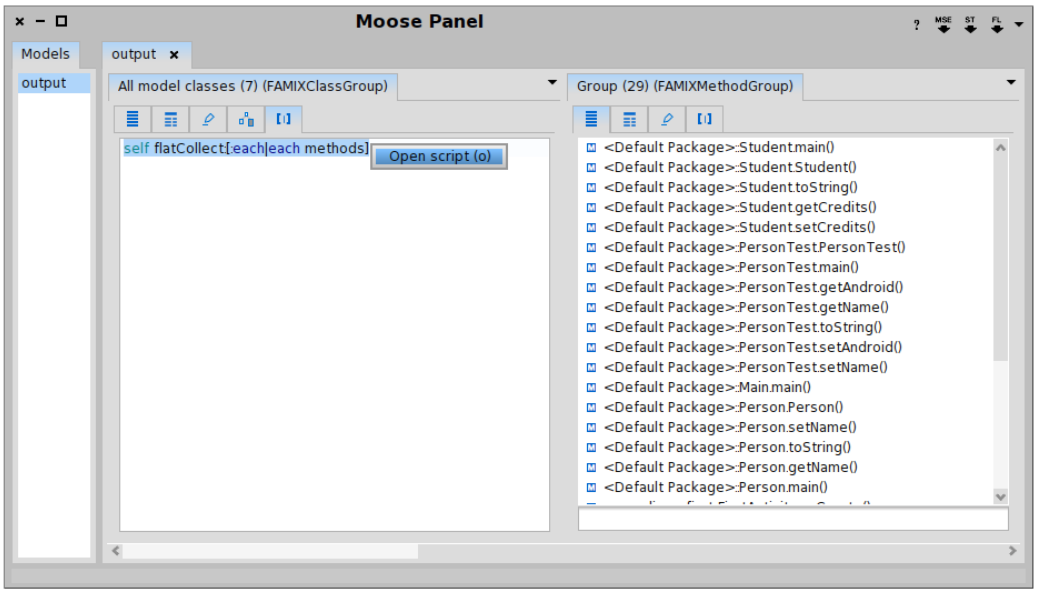
- Selection is supported through the interface dedicated to a group. By entering a query in the bottom text box, the corresponding entities are selected in the group and this selection can consequently be seen on the next pane.



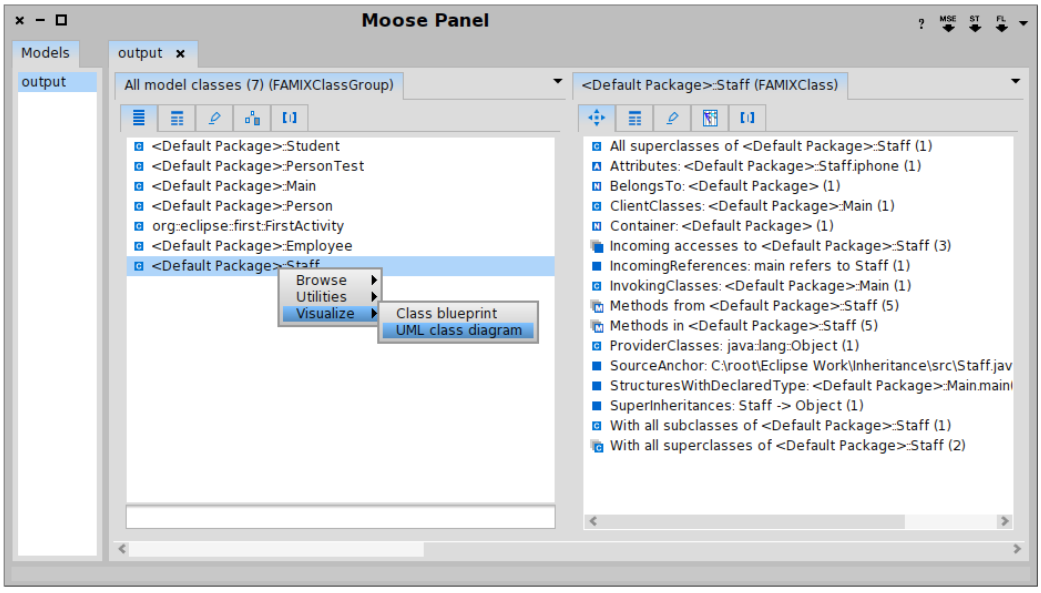
- Navigation can be achieved through focused on a single entity; the Finder offers a Navigation pane with a list of possible entities that can be spawned from the current entity.

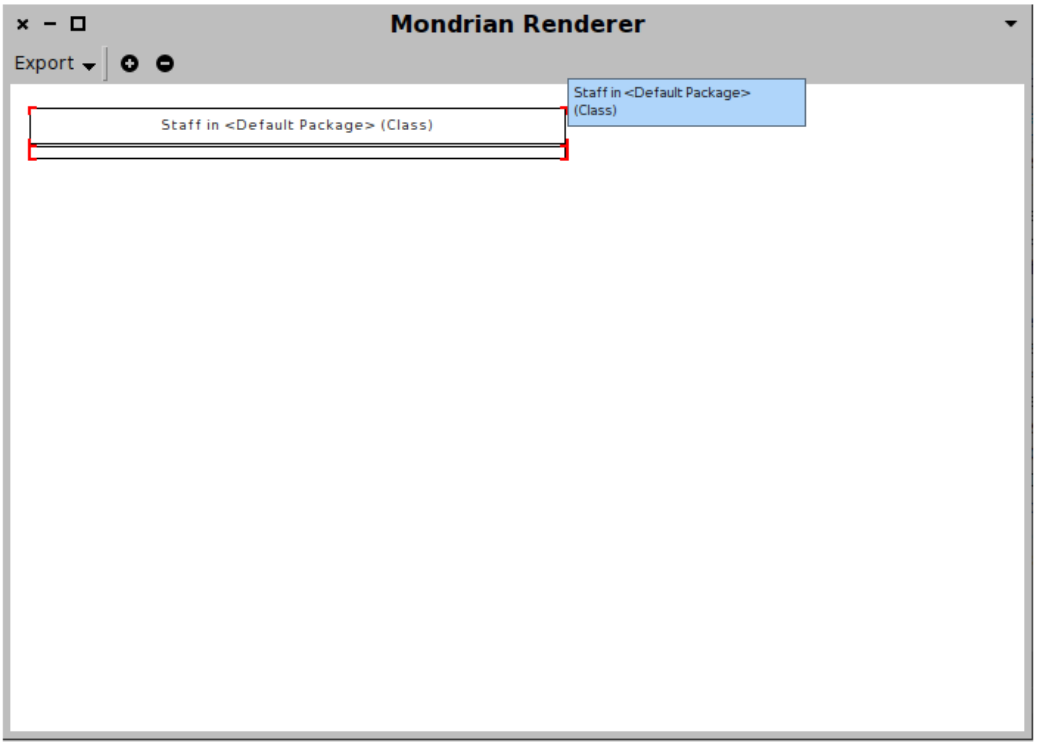


- Sometimes, the default navigation paths are not enough. For example, given a class we might want to get to all methods from the system that reference this class.
- When we want to obtain information from a group, for example, from a group of classes we might want to obtain all methods. To accommodate these cases the Finder offers a workspace in the Evaluator tab. In this workspace we have access to the current entity via self and we can evaluate any Smalltalk code.
- In the picture below we see the result of obtaining all methods from a group of classes.

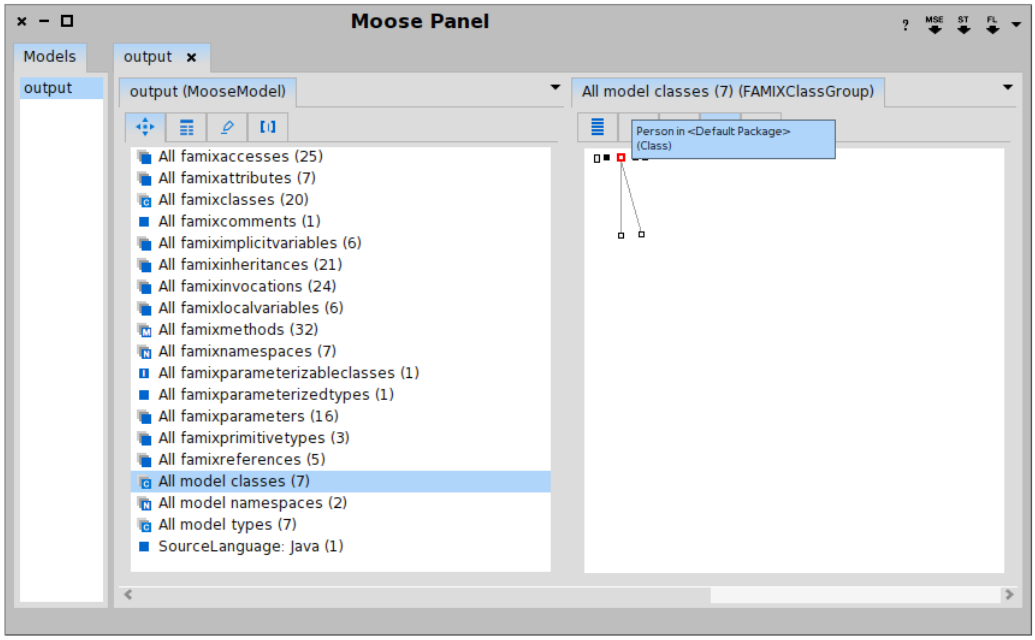


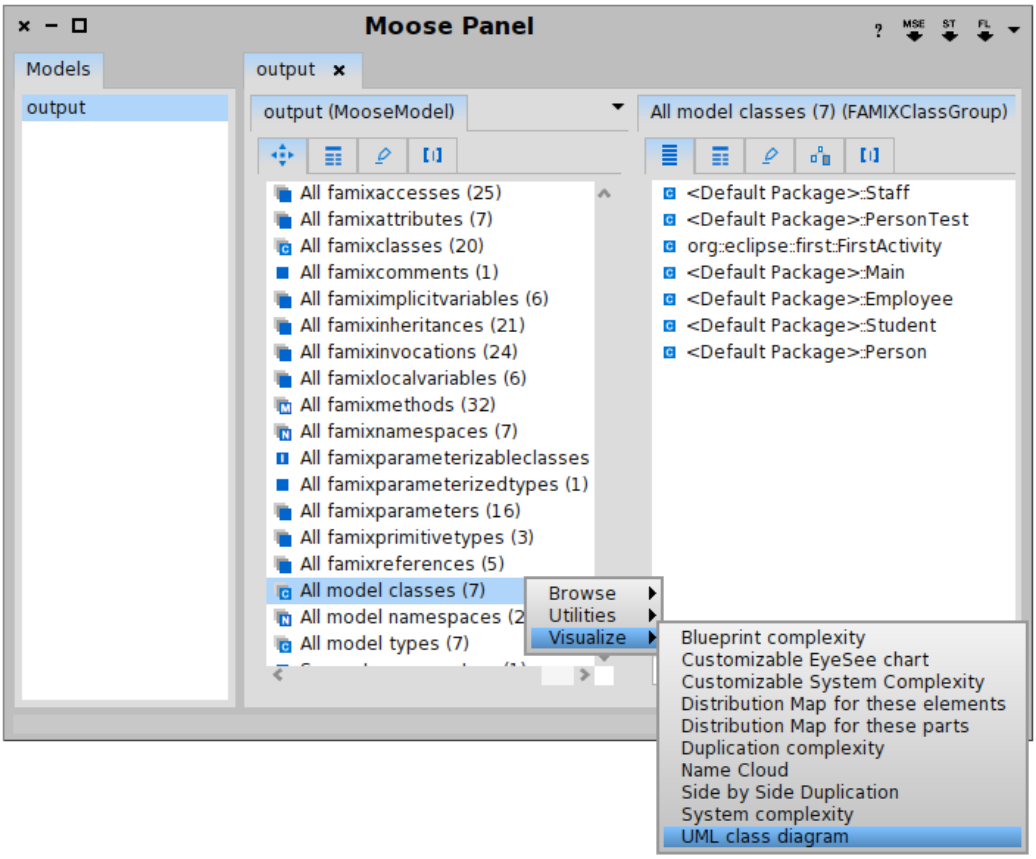
- Presentation is supported in multiple ways. First, for each entity, there are several tabs available.

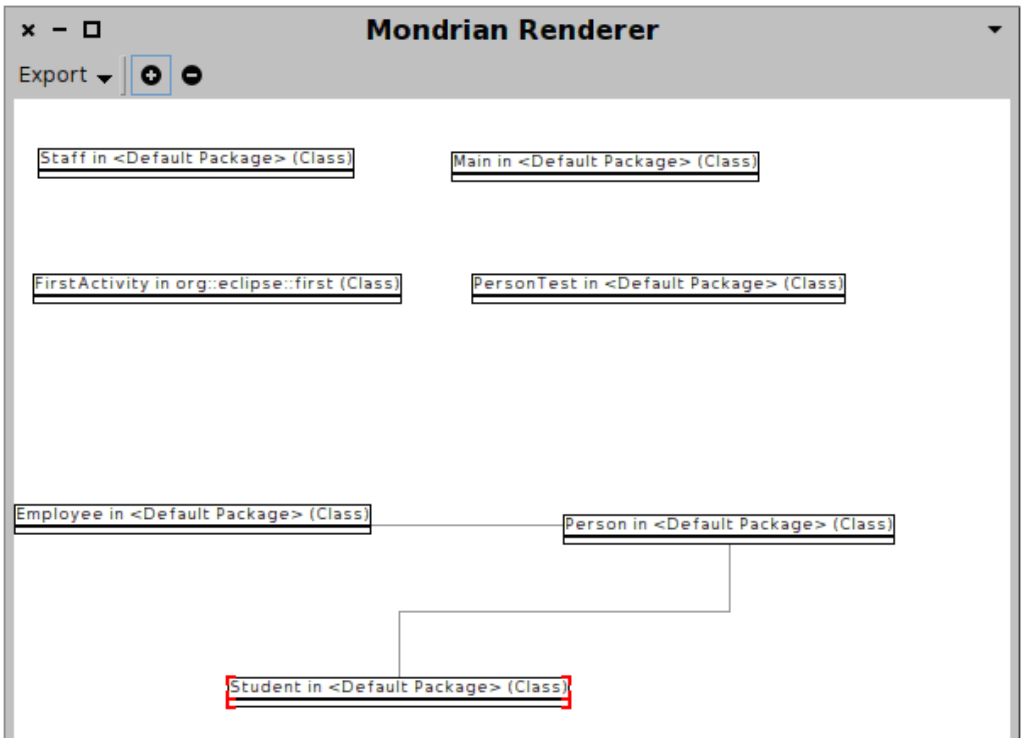




- Presentation, for each entity, Moose offers a menu through which further visualizations and tools can be spawned.




Moose Panel
Models
output
output (MooseModel)
All model classes (7) (FAMIXClassGroup)
All famixaccesses (25)
All famixattributes (7)
All famixclasses (20)
All famixcomments (1)
All famiximplicitvariables (6)
All famixinheritances (21)
All famixinvocations (24)
All famixlocalvariables (6)
All famixmethods (32)
All famixnamespaces (7)
All famixparameterizableclasses
All famixparameterizedtypes (1)
All famixparameters (16)
All famixprimitivetypes (3)
All famixreferences (5)
All model classes (7)
All model namespaces (2
All model types (7)
<Default Package>::Staff
<Default Package>::PersonTest
org::eclipse::first::FirstActivity
<Default Package>::Main
<Default Package>::Employee
<Default Package>::Student
<Default Package>::Person
Browse
Utilities
Visualize
Blueprint complexity
Customizable EyeSee chart
Customizable System Complexity
Distribution Map for these elements
Distribution Map for these parts
Duplication complexity
Name Cloud
Side by Side Duplication
System complexity
UML class diagram

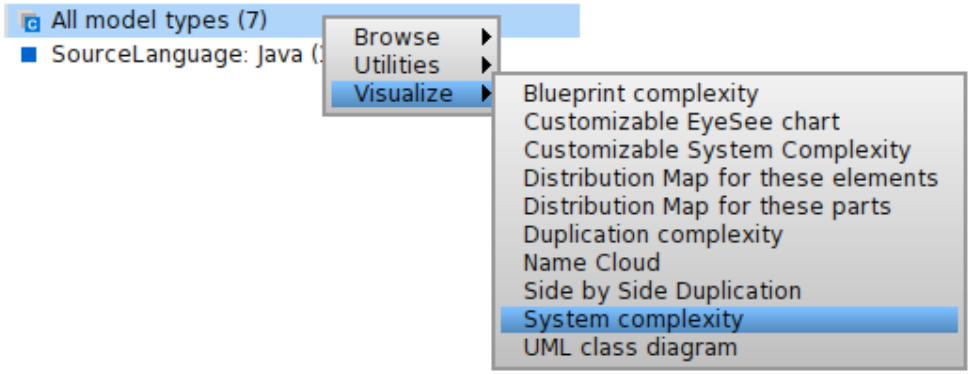

Mondrian Renderer
Export
Staff in <Default Package> (Class)
Main in <Default Package> (Class)
FirstActivity in org::eclipse::first (Class)
PersonTest in <Default Package> (Class)
Employee in <Default Package> (Class)
Person in <Default Package> (Class)
Student in <Default Package> (Class)

- System Complexity:

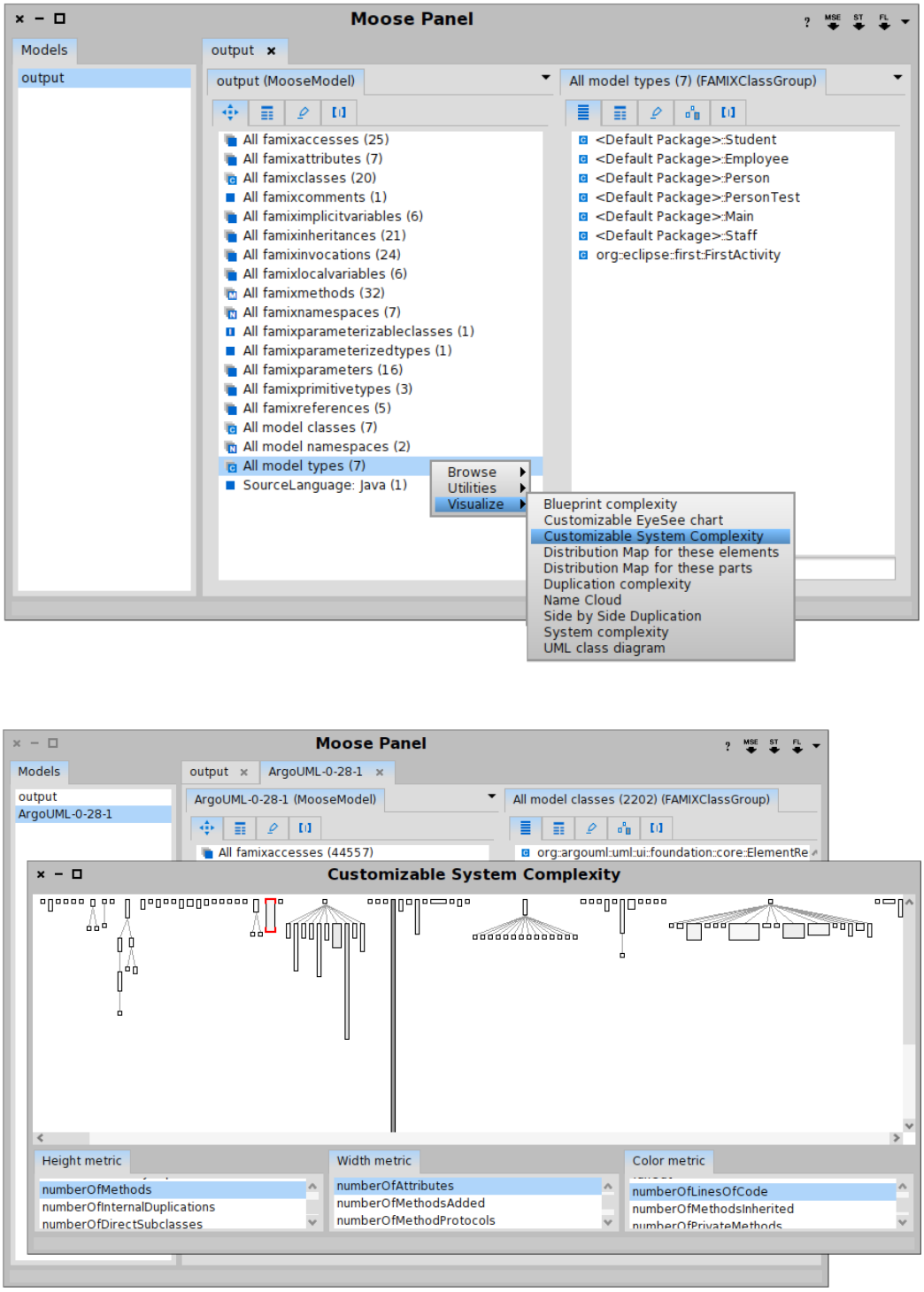
  - The classic System Complexity is a polymetric view showing a graph in which nodes represent classes and edges represent inheritance relationships between classes. The nodes are visually enriched with three metrics [Height, Width, and Color metric]:

    - The number of methods (NOM) is mapped on the Height metric.
    - The number of attributes (NOA) is mapped on the Width metric.
    - The number of lines of code (LOC) is mapped on the Color metric.

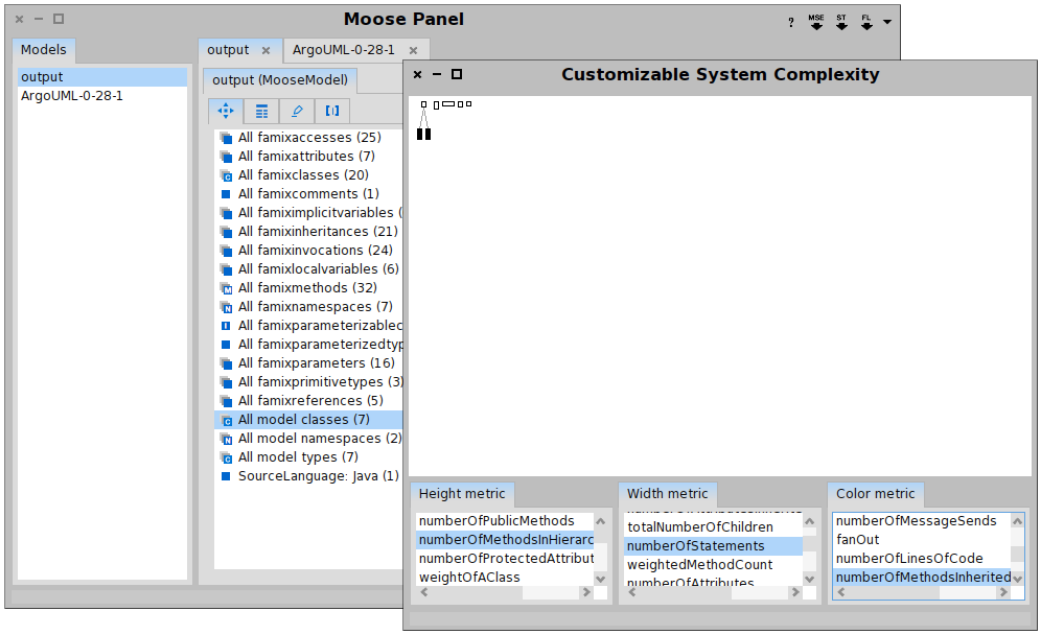
  - You can obtain the default visualization through the contextual menu of any class group:



  - You can also get a user interface that enables you to select the properties that you want mapped on each dimension:



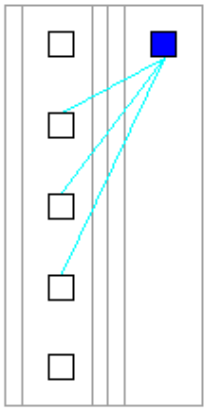
[Customizable System Complexity]



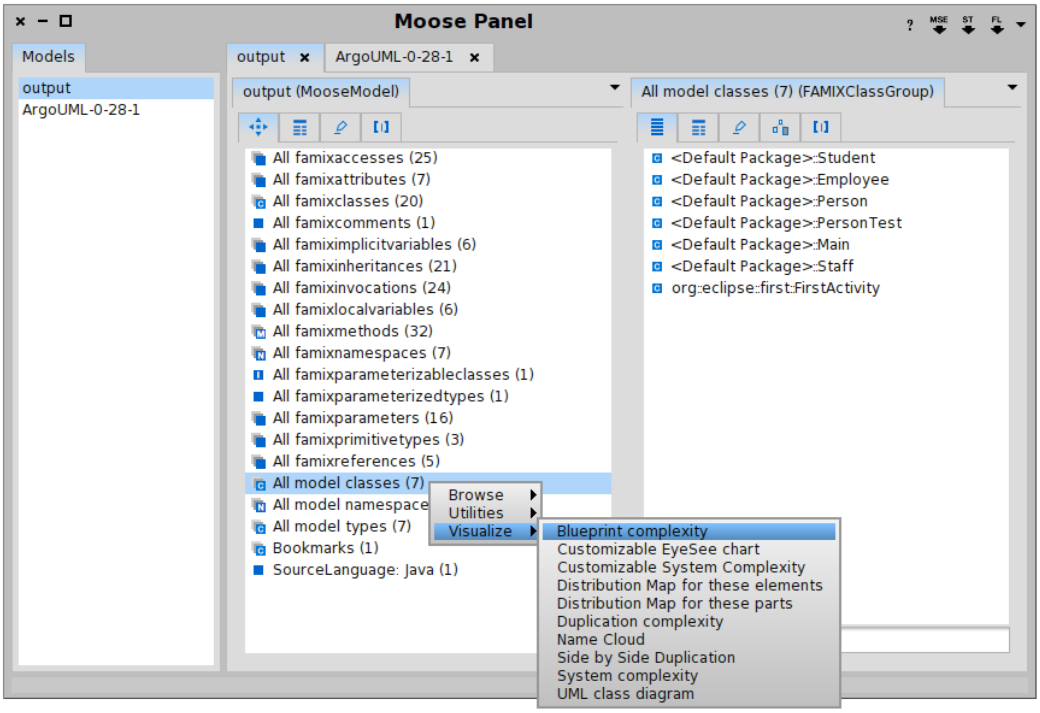
[Customizable System Complexity]

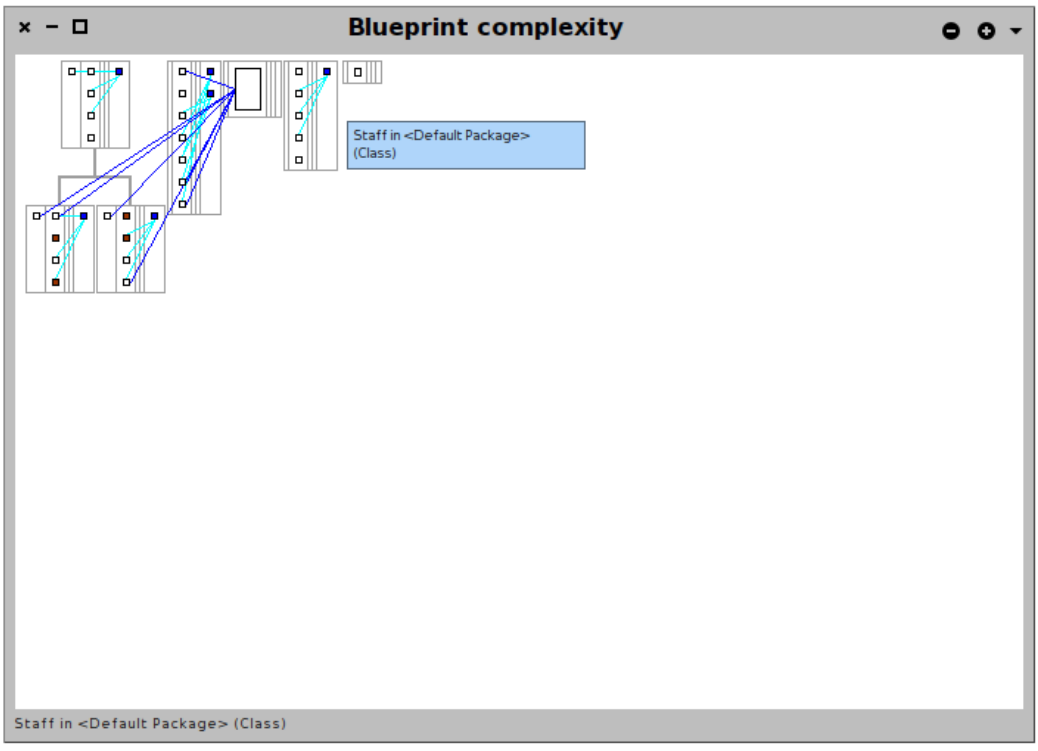
- Class Blueprint:

o The Class Blueprint is another famous polymetric view that shows the internals of a class. The class is split into 5 layers: [Initialization, Public Interface, Private Implementation, Accessor, and Attribute layer].



o Furthermore, there are two types of edges: [1: Invocations between methods (shown with blue). 2: Accesses from methods to attributes (shown with cyan)].

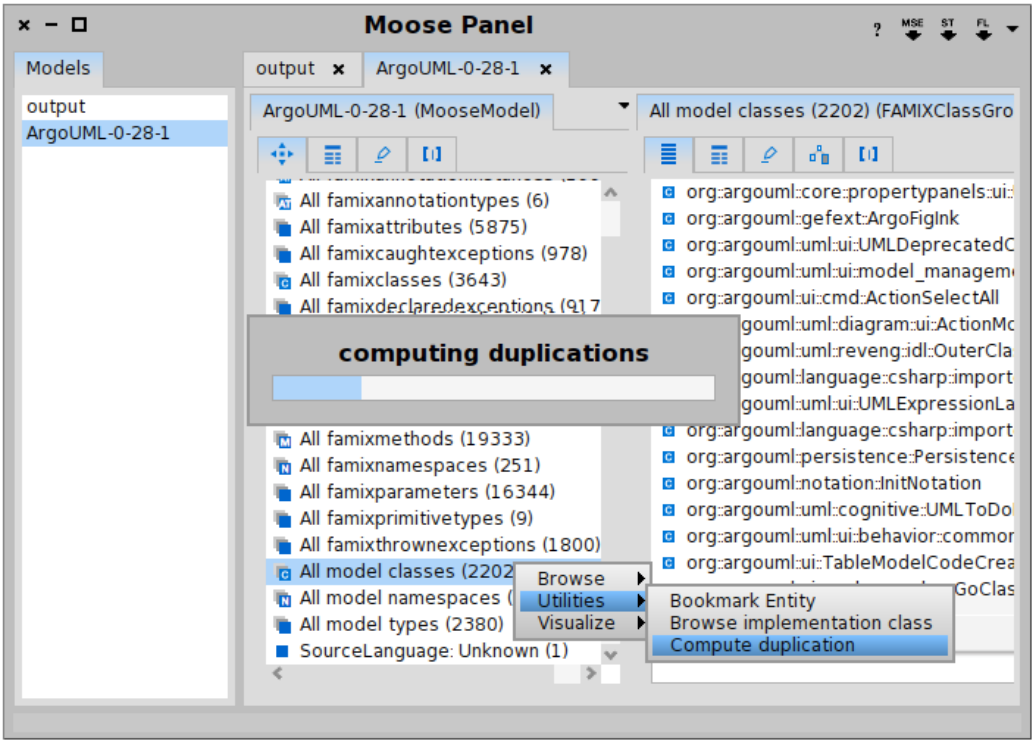


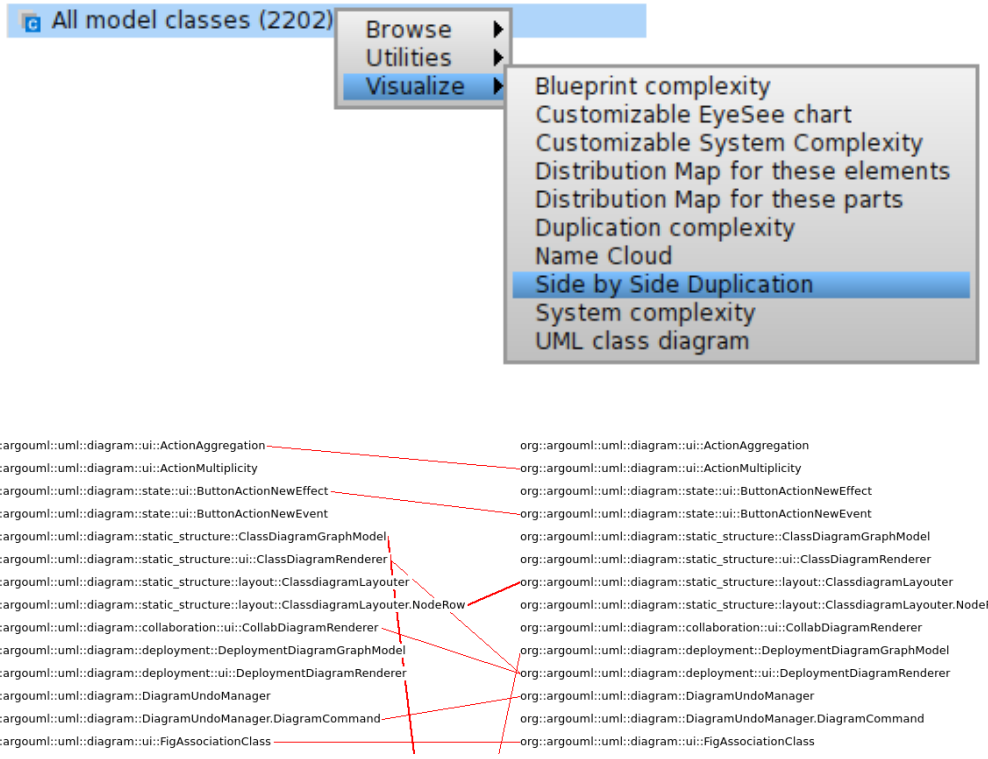


[Blueprint complexity]

- Side-by-side duplications:

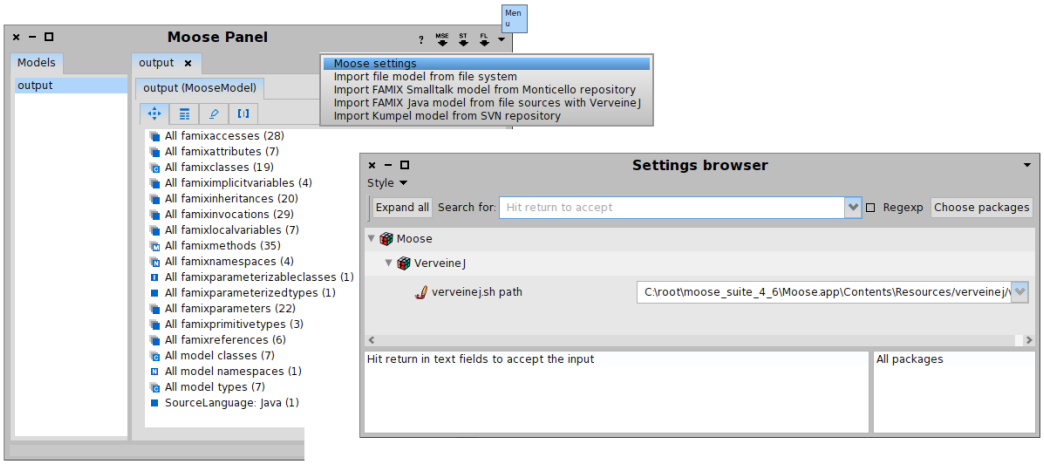
  - The visualization called Side-by-side Duplications shows the classes, methods, and files that are involved in duplications. They organized into two columns both showing the same entities. The red edges show the duplications, and the width of these edges indicates the number of duplicated fragments between the corresponding classes.
  - This visualization can be spawned by first computing duplications for classes, and then by executing Visualize/Side-by-side duplications from the contextual menu of a group of classes.





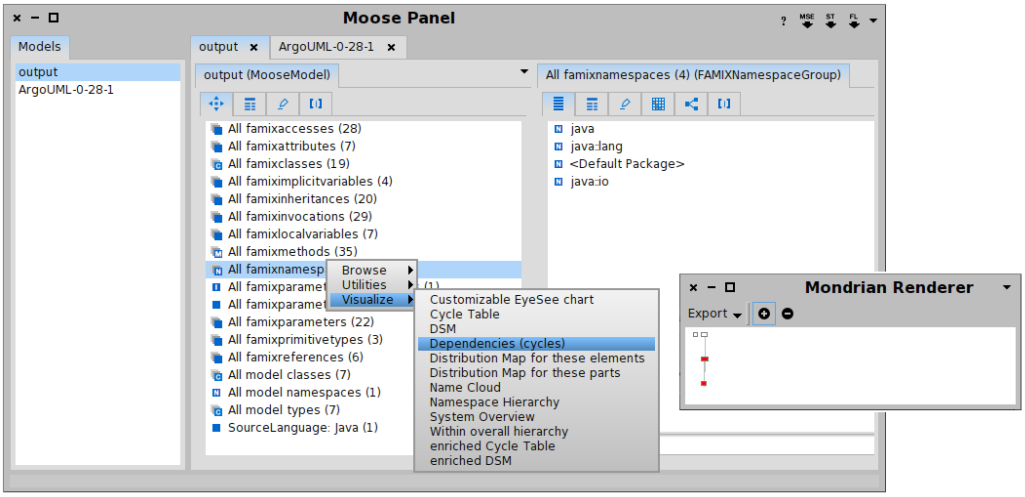
[Side-by-side duplication of the classes]

- Setting the Root Folder:



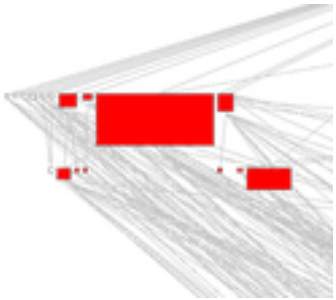
- Namespace and package dependencies:

Namespace Dependencies and Package Dependencies are polymetric views in which each node is a namespace or package and each edge represents a dependency.



[Example of the Namespace Dependencies with Cycles visualization]

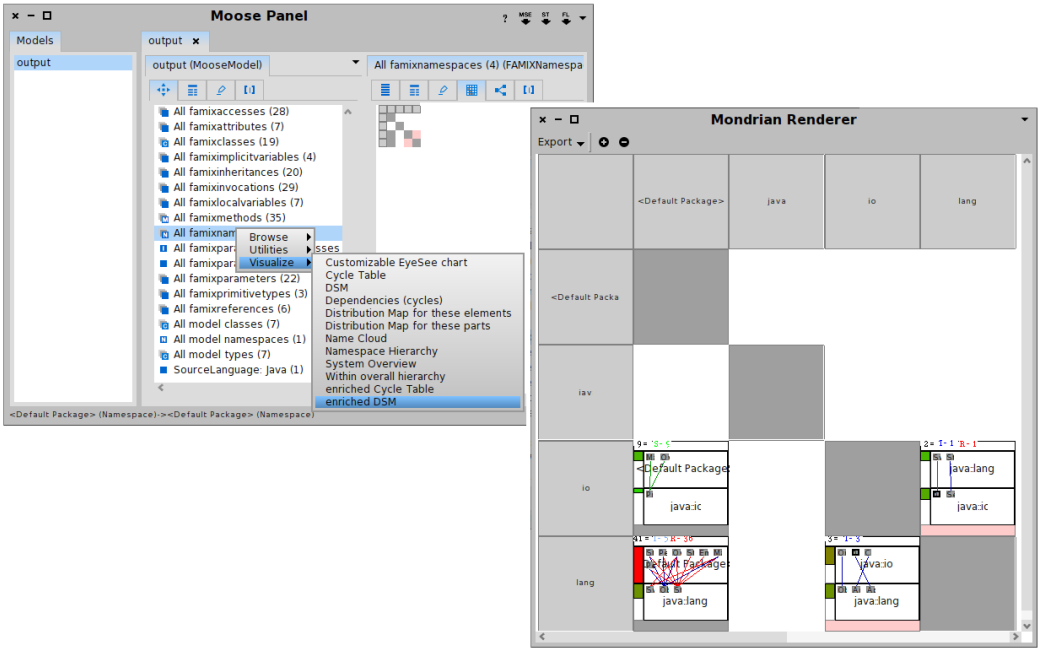
The nodes are enhanced with the following metrics: [1: The Width is given by the number of classes (NOCls) in the namespace or package, and [2] The Height is provided by the number of methods (NOM) in the namespace or package].



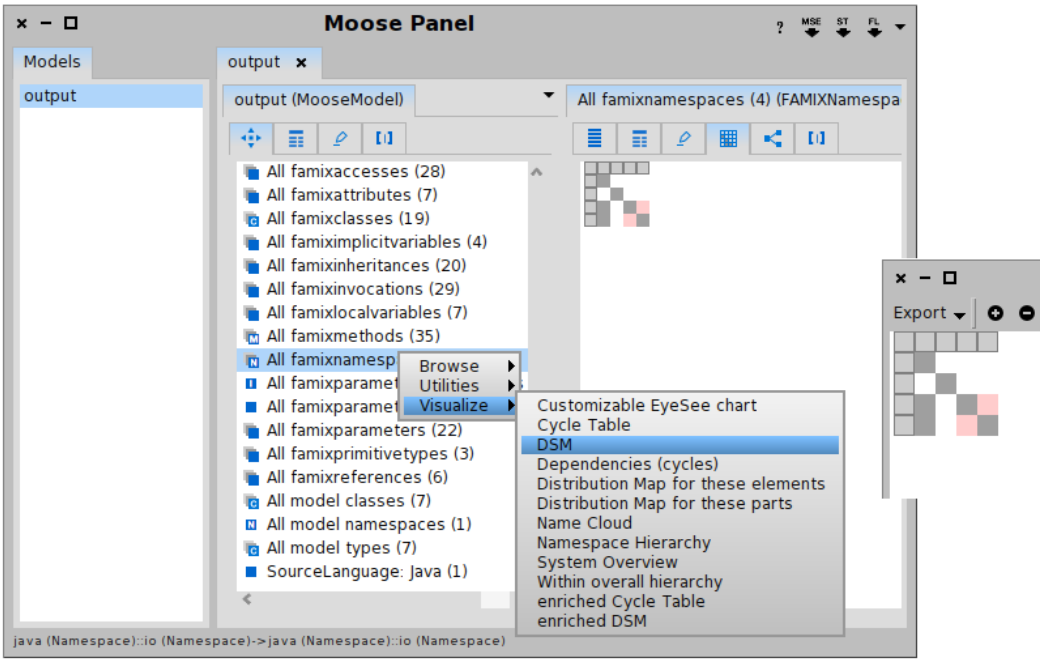
- Enriched Dependencies Structure Matrix:

- The Enriched Dependencies Structure Matrix (EDSM) shows the dependencies between parts (e.g., namespaces or packages) and highlights dependency cycles.
- The matrix displays the same parts both on columns and on rows, and each dot in the matrix denotes a dependency.
- The algorithm tries to arrange the parts so that all dependencies are below the main diagonal. Thus, the dots that are above the diagonal introduce cyclic

dependencies. There are red, pink and yellow dots each of these indicating a dependency that is involved in a cycle.



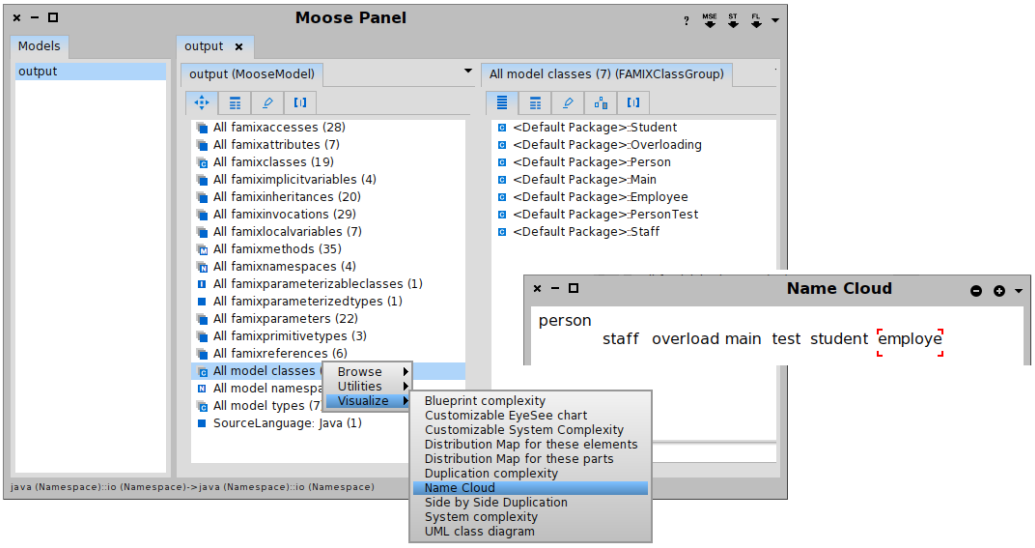
[Enriched DSM example]



[DSM example]

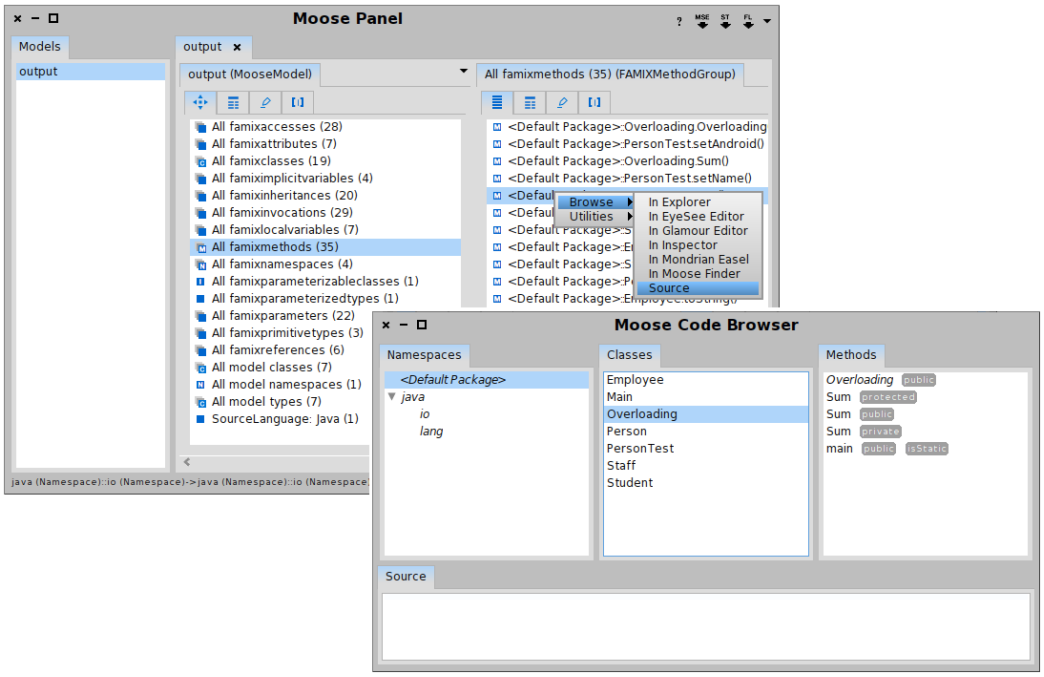
- Name Cloud:

  - Name cloud simply provides a means to identify the words used in the names of entities from a group. Its goal is to provide a quick preview of the vocabulary used in the items from the group.
  - You can invoke the visualization on any group from the contextual menu: Visualize / Name Cloud.

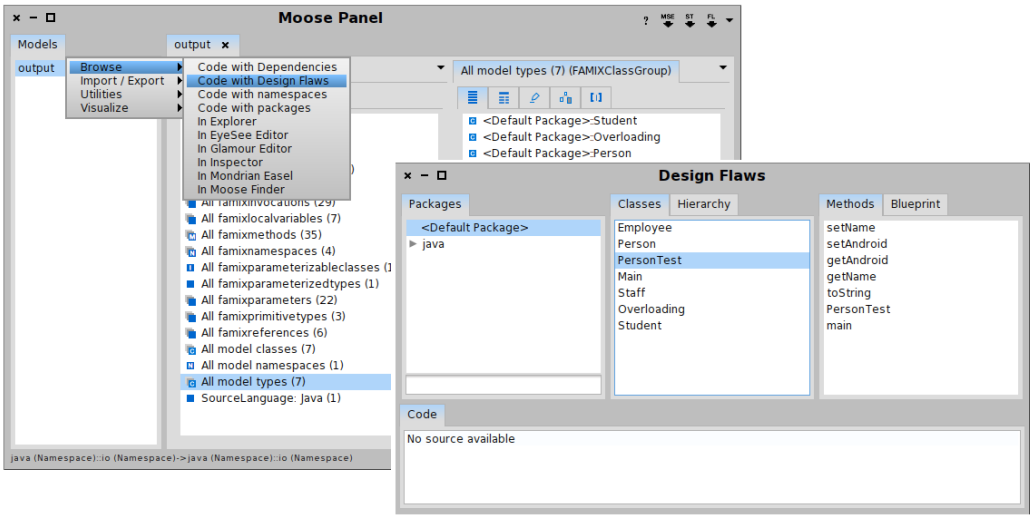


[Name cloud of all classes]

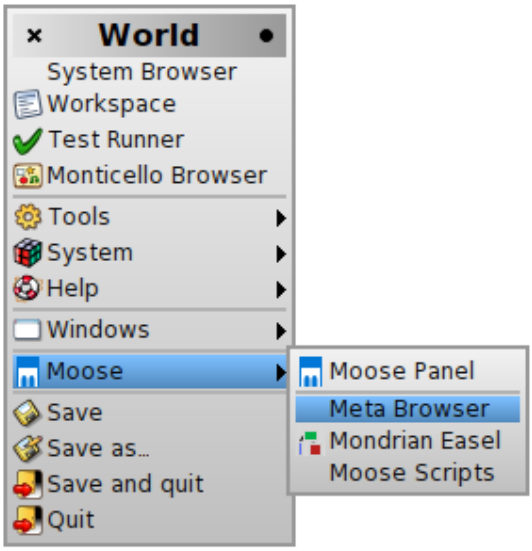
- Moose Code Browser:

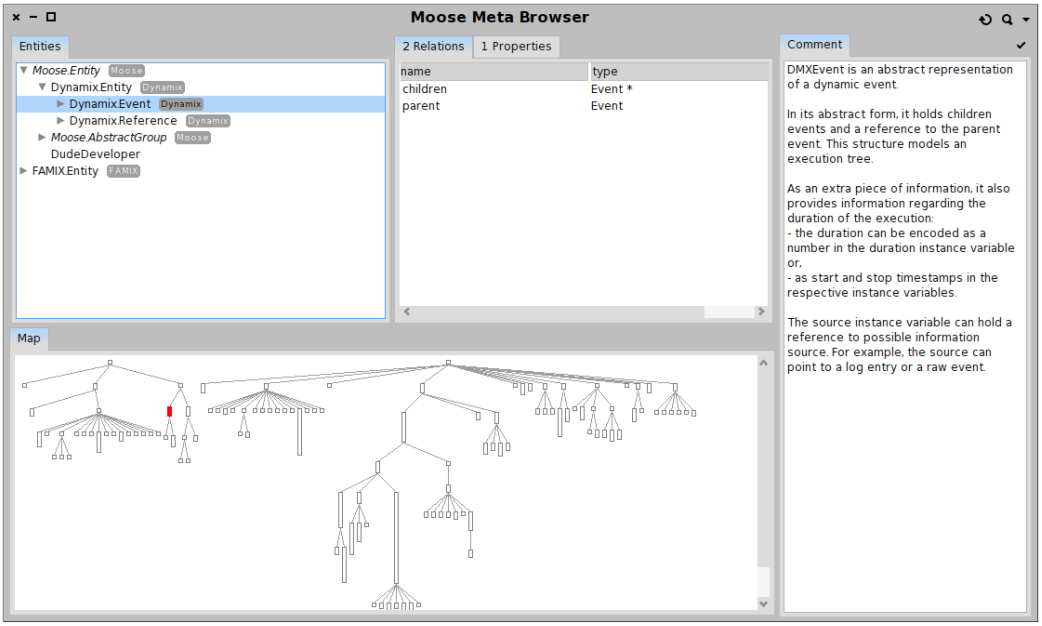


- Moose Browse:



- Moose Browse:

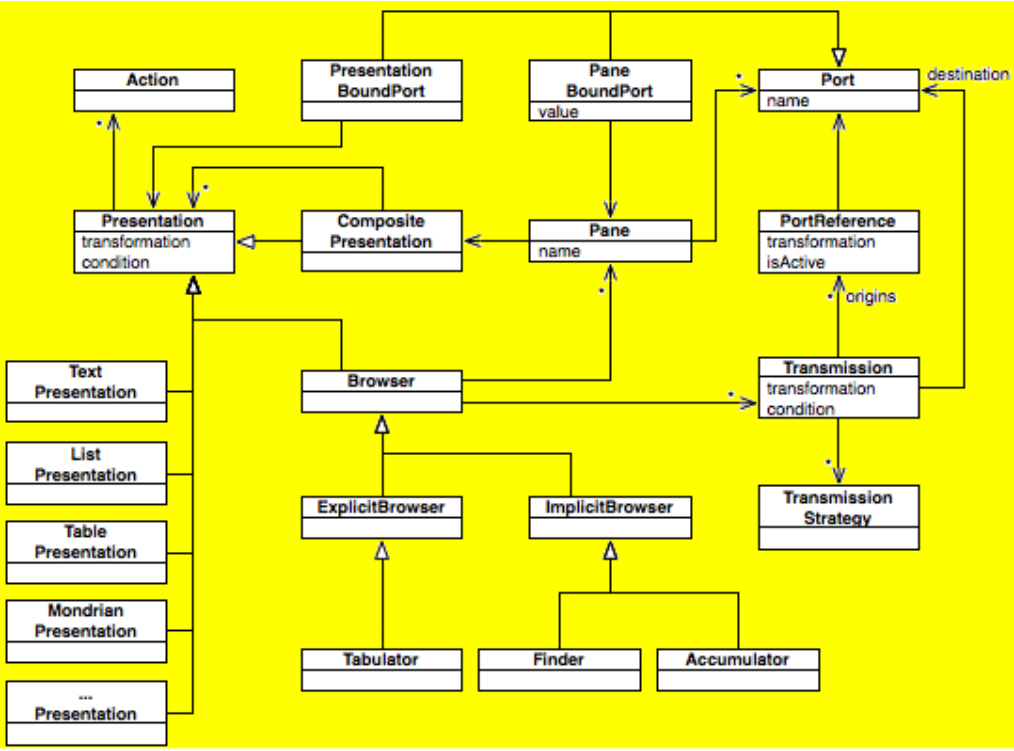




[Moose Meta Browser]

- Glamour architecture:

The architecture of Glamour revolves around a handful of concepts. The picture below shows a UML diagram with an overview of these.



[Glamour Architecture]

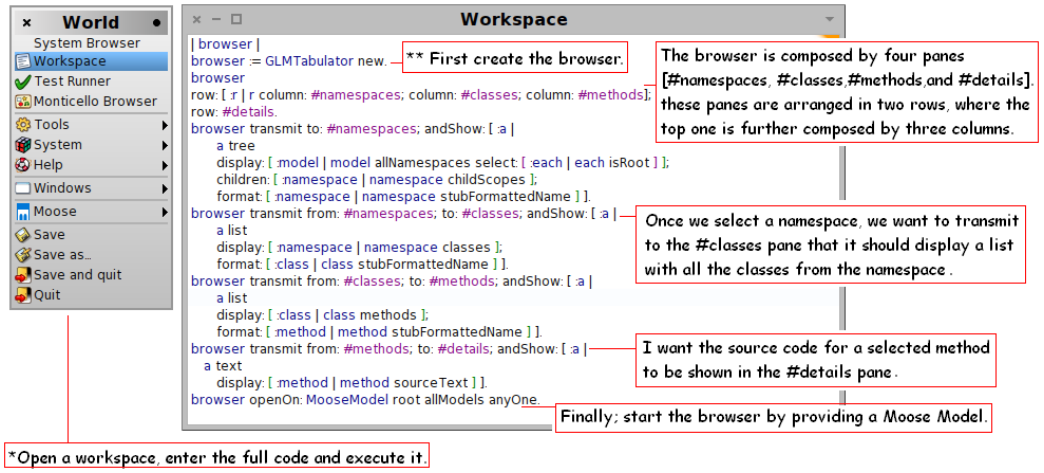
- The central concept is the Browser.
- A Browser is composed of Panes.
- The navigation flow is specified via Transmissions.
- A Transmission connects several origin ports with a destination port, and its job is to handle the propagation of values between these.
- Panes only define the location in the Browser, but the way they should be rendered is handled by Presentations.
- A Presentation interprets the state of the Pane and maps it to visual characteristics through a strategy pattern. Thus, we can have multiple types of Presentations: Text presentation, List presentation, Table presentation, Mondrian Presentation, and so on.

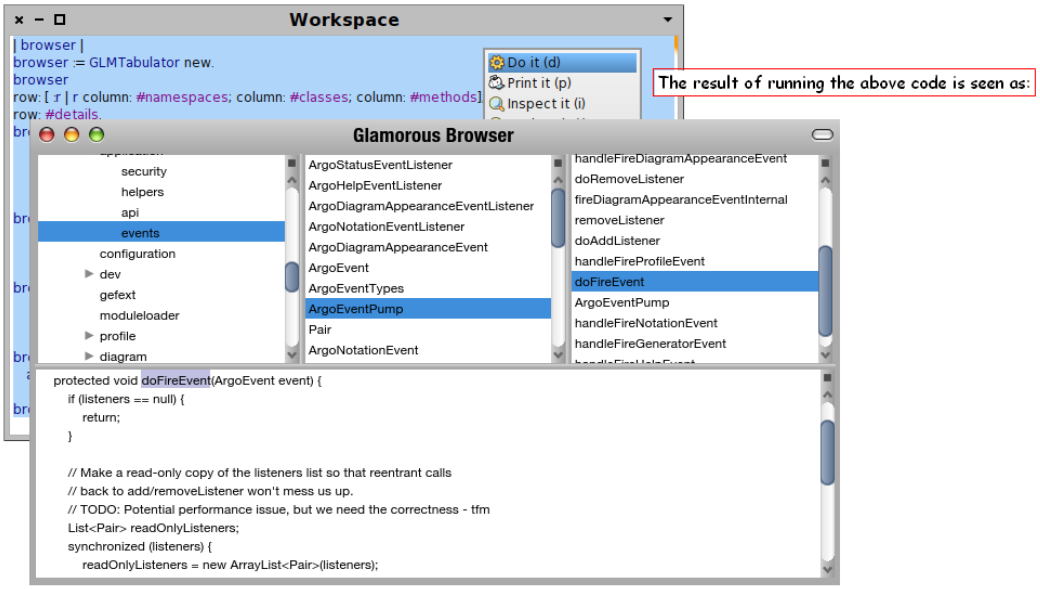
A browser [1] is made of panes; [2] which represent spatial locations on which various objects [3] are presented. [4] The result of acting on the graphical presentation is transmitted [5] to other panes.

- Building browsers with Glamour:

A browser is a specific user interface that allows us to look at the space provided by the model, to navigate from one part of this space to another, and to act upon it.

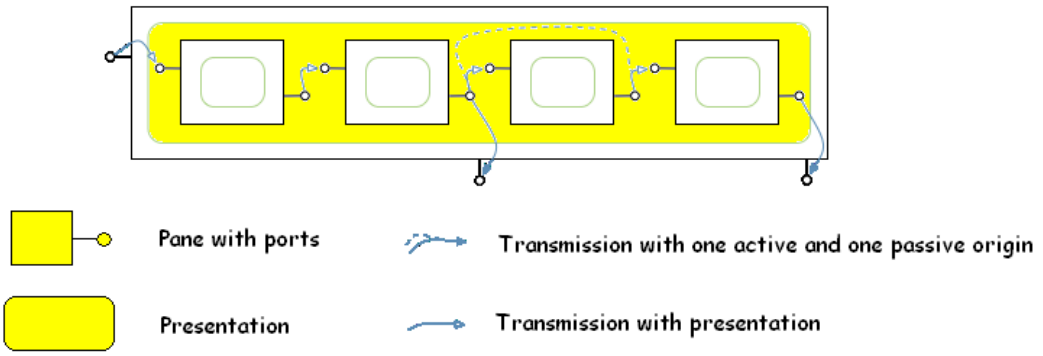
- A glimpse of Glamour:





- Sketching browsers:

To ease the conception of browsers we can use a dedicated notation to capture the critical aspects: Pane, Port, Transmission and Presentation (and Browsers).



This notation is meant to be used during the process of designing the browser. Use this even for simple browsers. Before coding a browser, start by sketching it on a paper.

- Handling transmissions:

The fundamental idea behind Glamour starts from the observation that the navigation flow should be orthogonal to the way of presentation. In Glamour, this navigation flow is captured through Panes, Ports and Transmissions: the Panes represent the building blocks, the Ports provide the hooks, and the Transmissions form the fiber of the browser by connecting the ports.

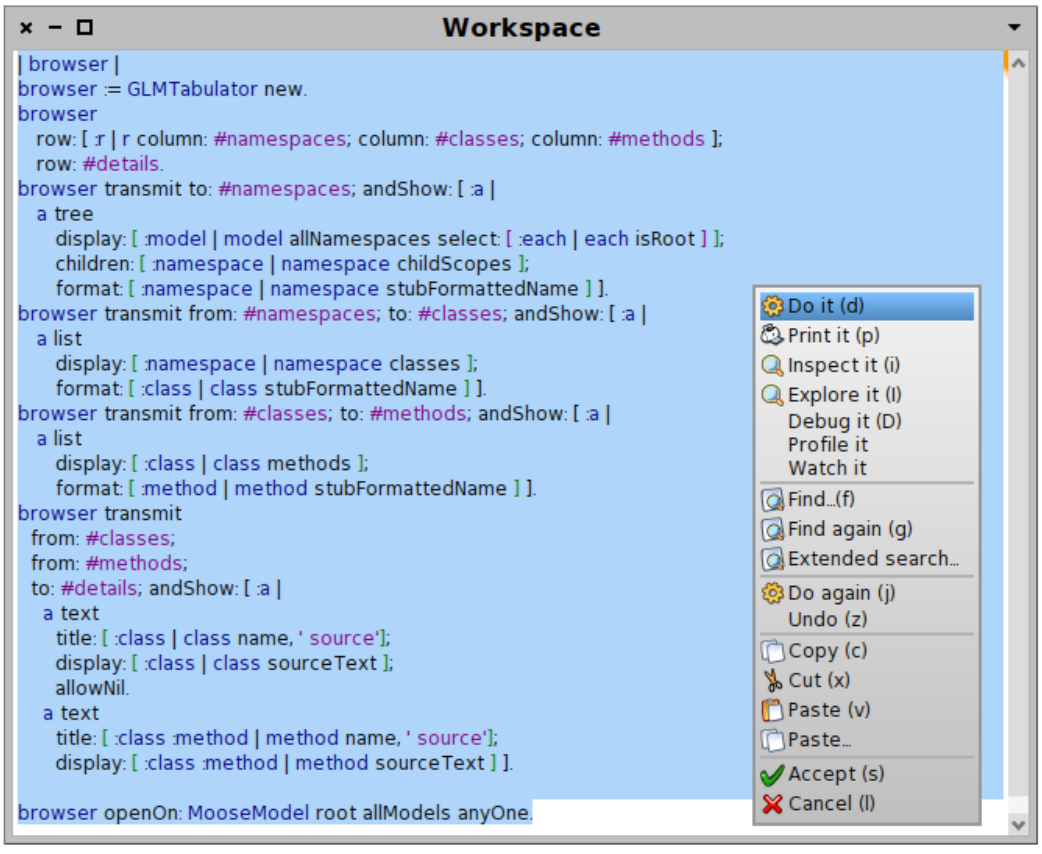
An origin port can be either active [specified using from: port:] or passive [specified using passively from: port:]. Only the active origin ports can trigger a transmission. This information is captured by the PortReference.

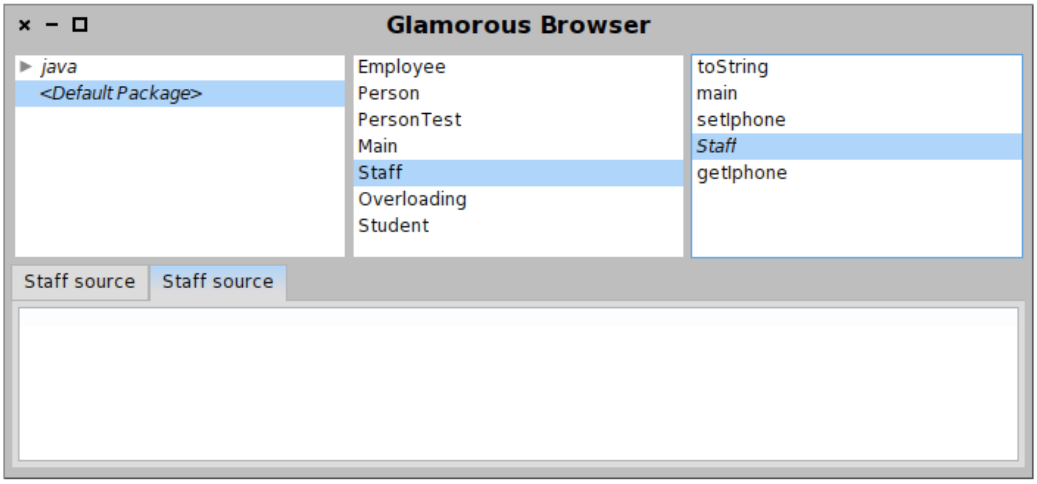
For specifying a transmission origin that points to a #selection port of a pane [from:].

Similarly, for specifying a transmission destination that points to #entity port of a pane [to:].

Presentations can be parameterized in several ways using blocks. The basic parameters are:

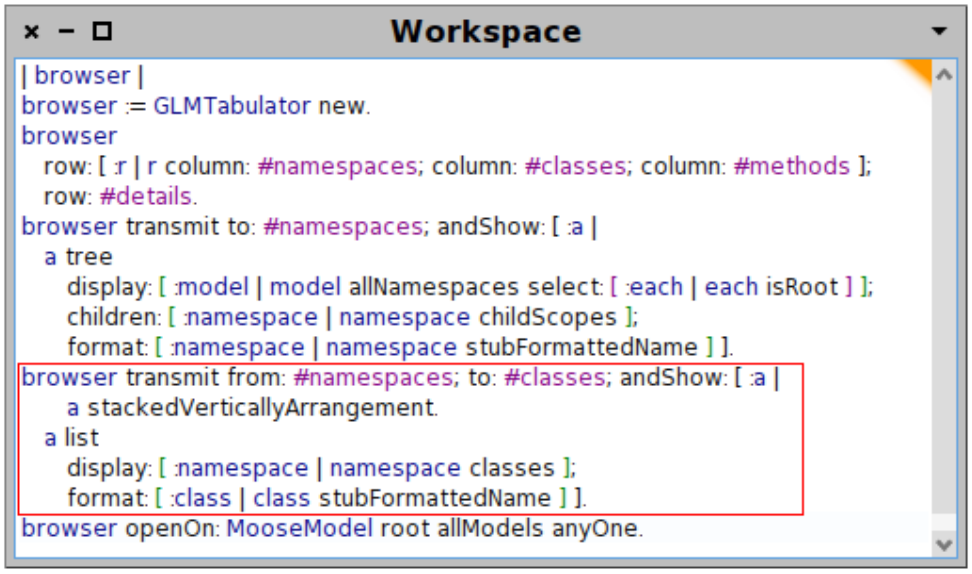
Every presentation has a title set via title: A presentation can transform the input into a model better suited for it. This is accomplished via the display: message. The visibility of each presentation can be controlled via a condition block that is set by sending a when: message. By default, there is an implicit condition saying that all input objects must be not nil for the presentation to appear. If we still want to allow the presentation to appear even if the input objects are nil we can use allowNil.





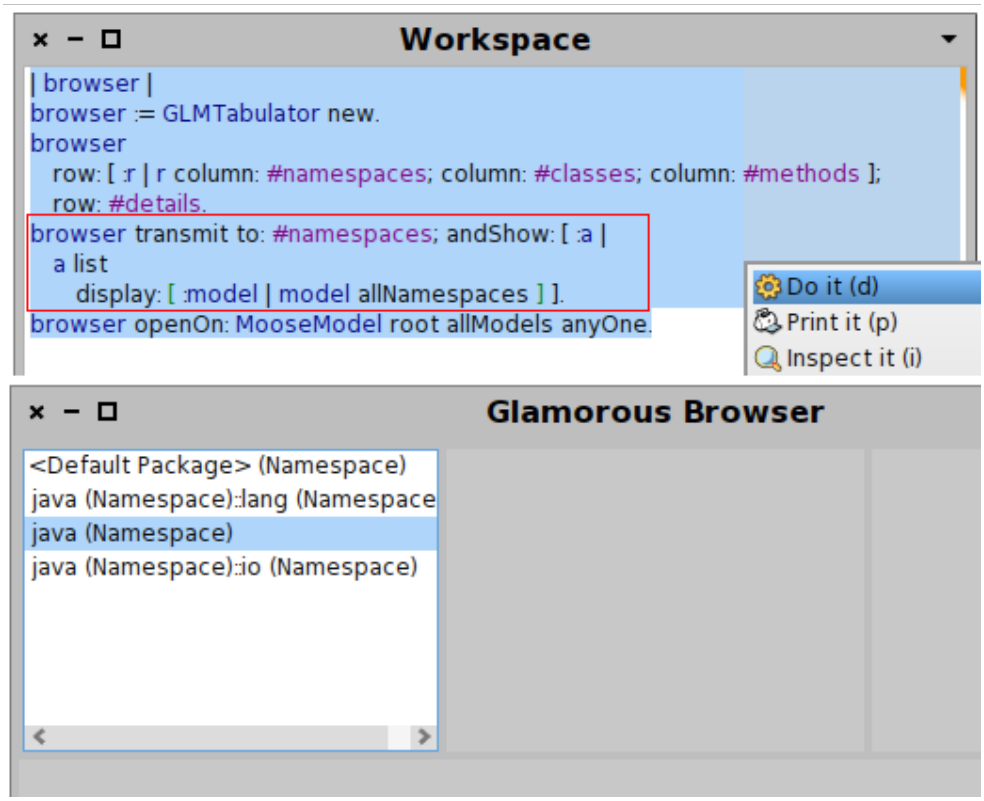
- Composite presentation:

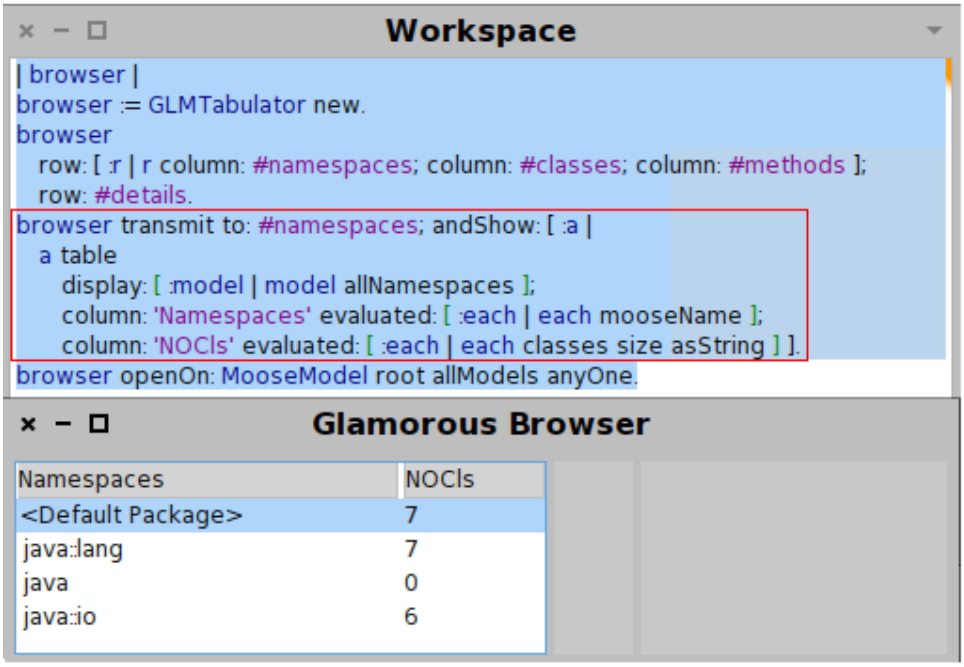
Every pane has a reference to a Composite Presentation. Thus, whenever we specify a transmission, we get access to such a composite presentation.



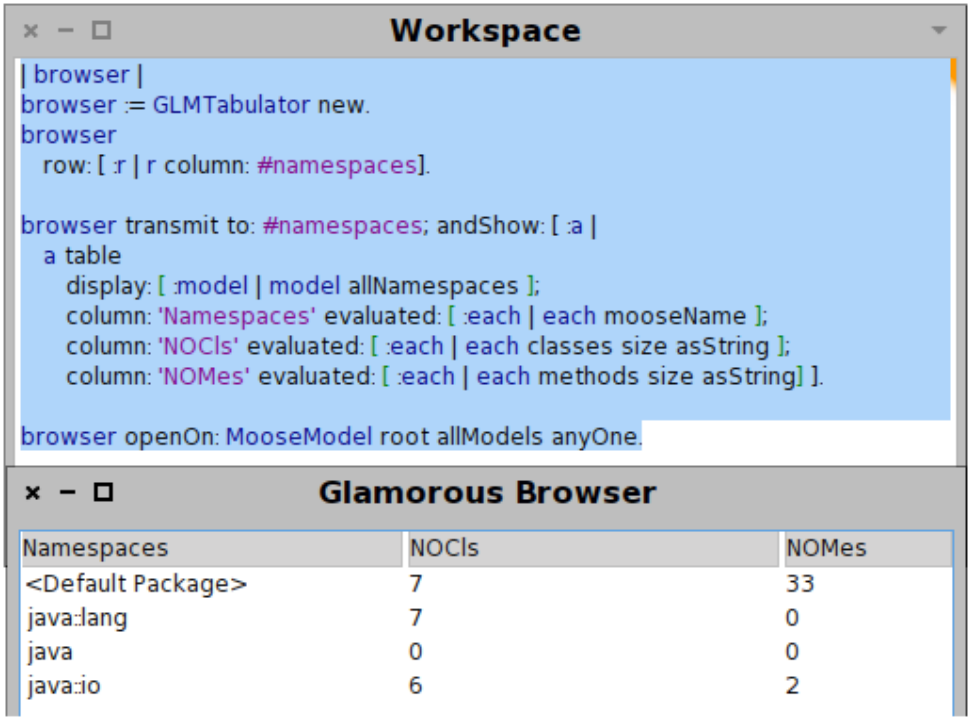
- Listing presentations:

There are three Listing Presentations available: List, Tree and Table. Each of these expects the result of evaluating the transformation on a given input entity [by using display:] to be a list.

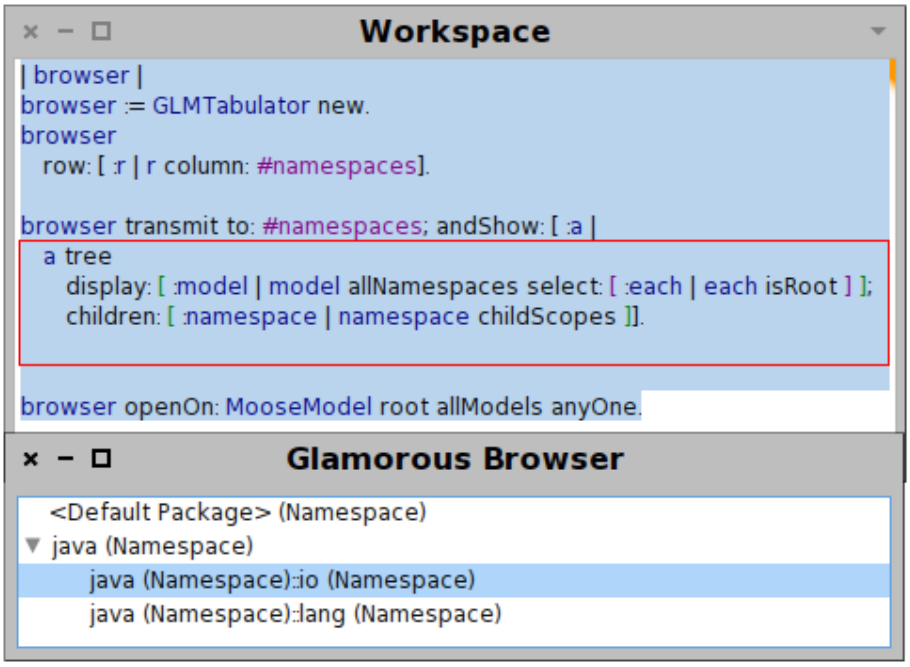




if we want to display the number of classes in each namespace, we can do it like this.



if we want to display the number of Methods in each class, we can do it like this.



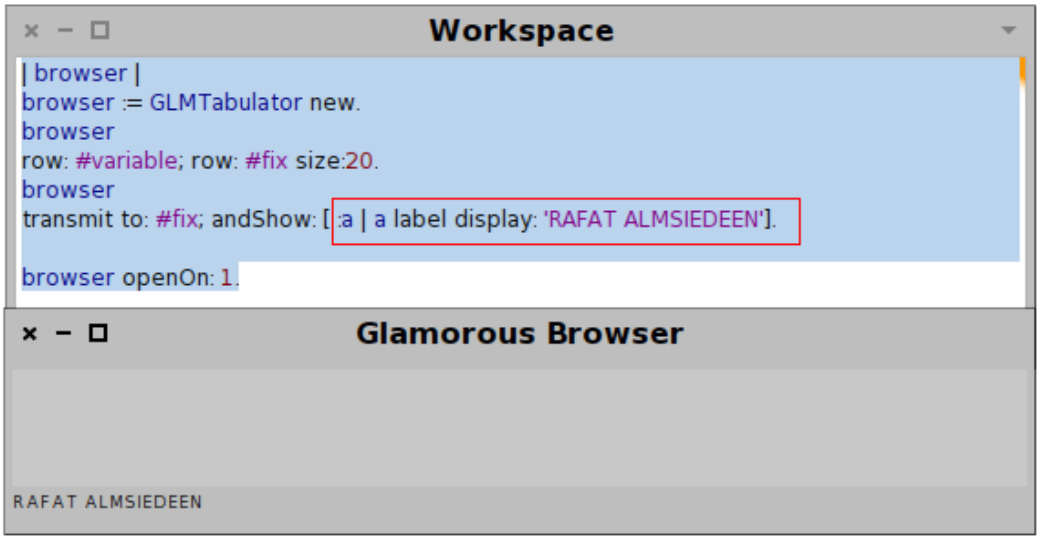
if we want to show the tree of all namespaces, in the display block we have to select only the root ones and then specify the children [using children:] for each of the namespaces:

- Text Presentation:

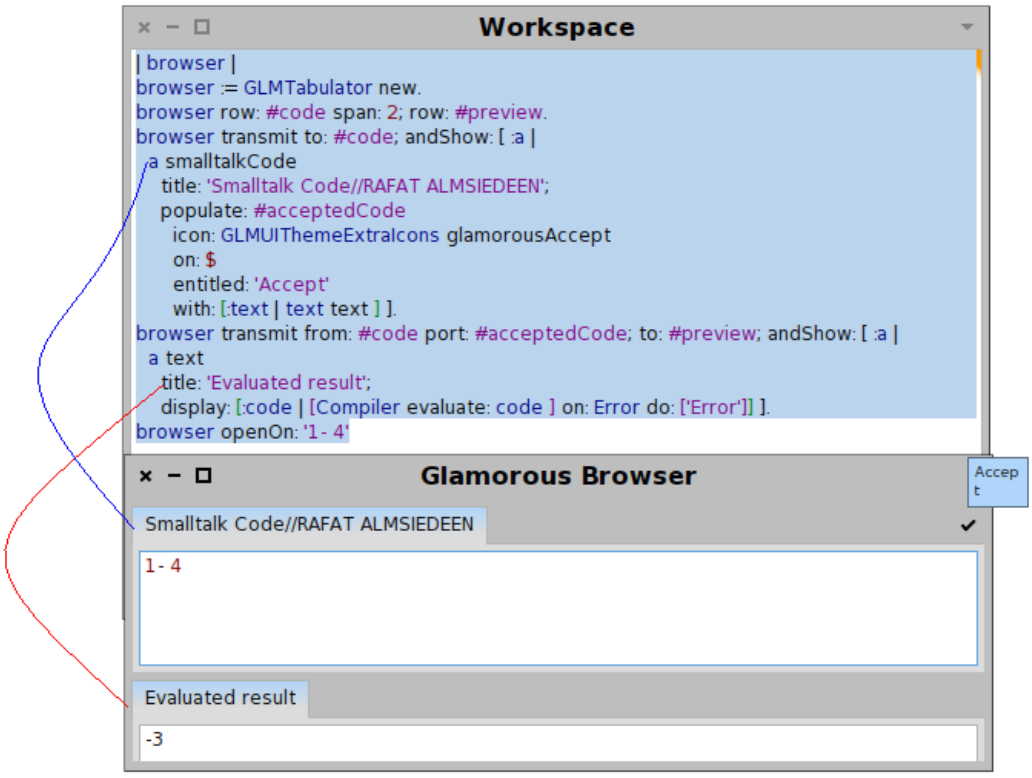
```
browser transmit from: #methods; to: #details; andShow: [ :a |
   a text
      display: [ :method | method sourceText ] ].
```

```
browser transmit to: #source; andShow: [ :a |
   a text
      display: [ :class | class sourceText ] ].
```

- Label Presentation:

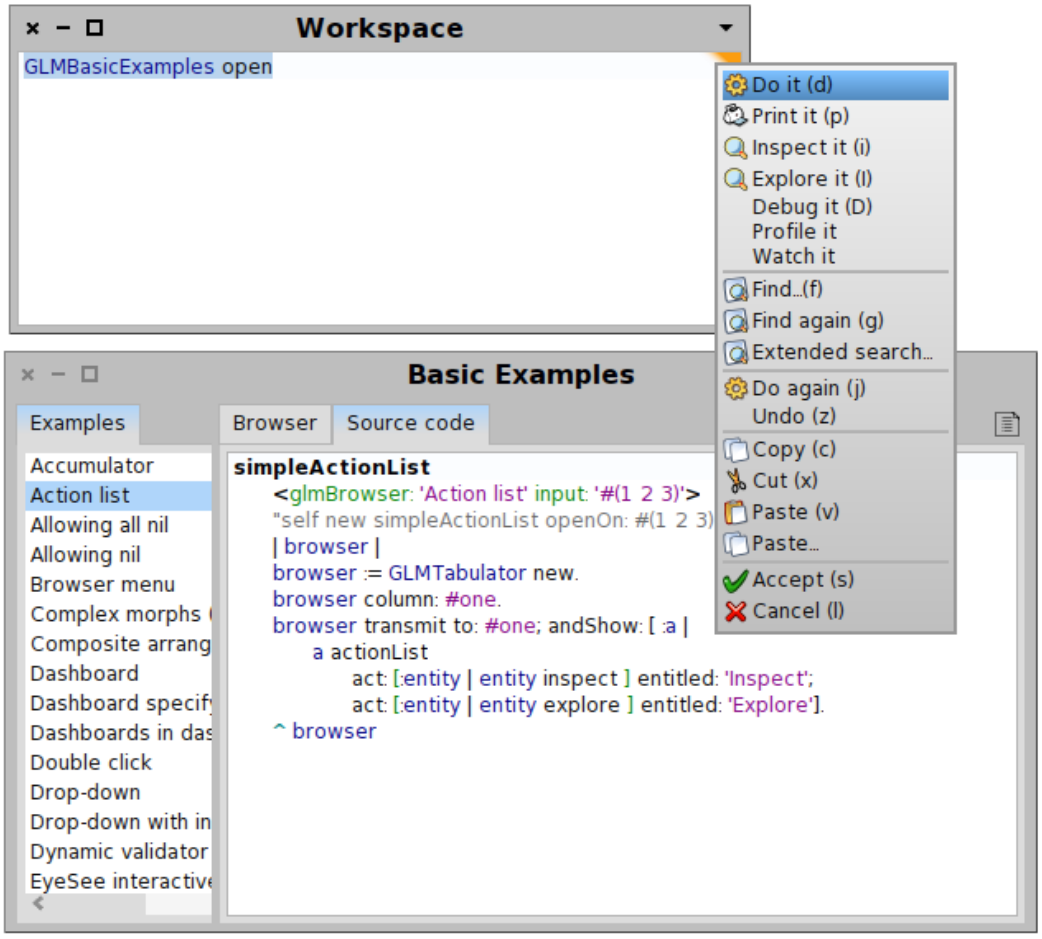


- Smalltalk code presentation:



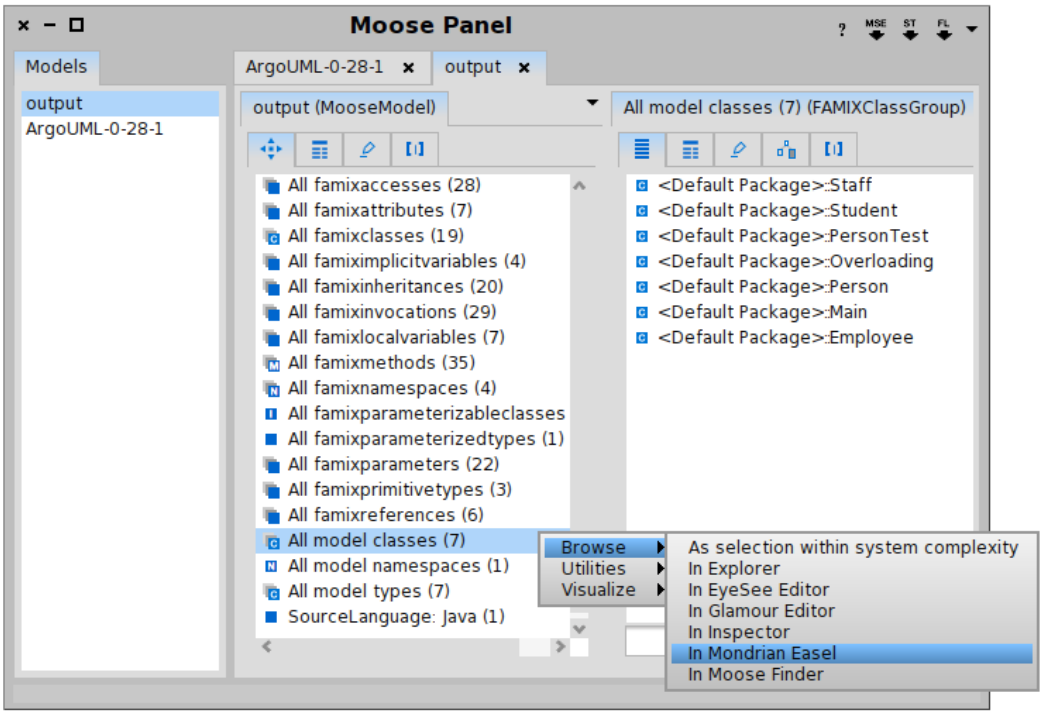
- Examples:

Glamour offers a self documenting browser with a set of examples covering the various features.

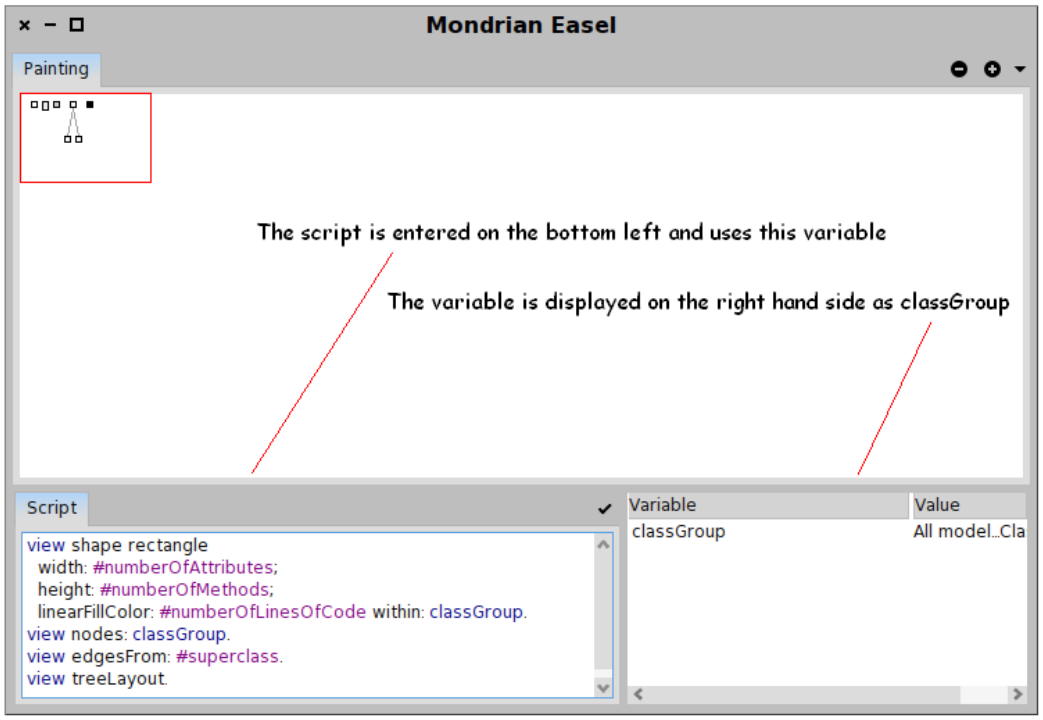


- Scripting in Mondrian Easel:

Mondrian provides the Mondrian Easel, a dedicated interface for scripting visualizations. The interface can be open on an input entity, and this entity can be directly accessed in the script via a variable name.



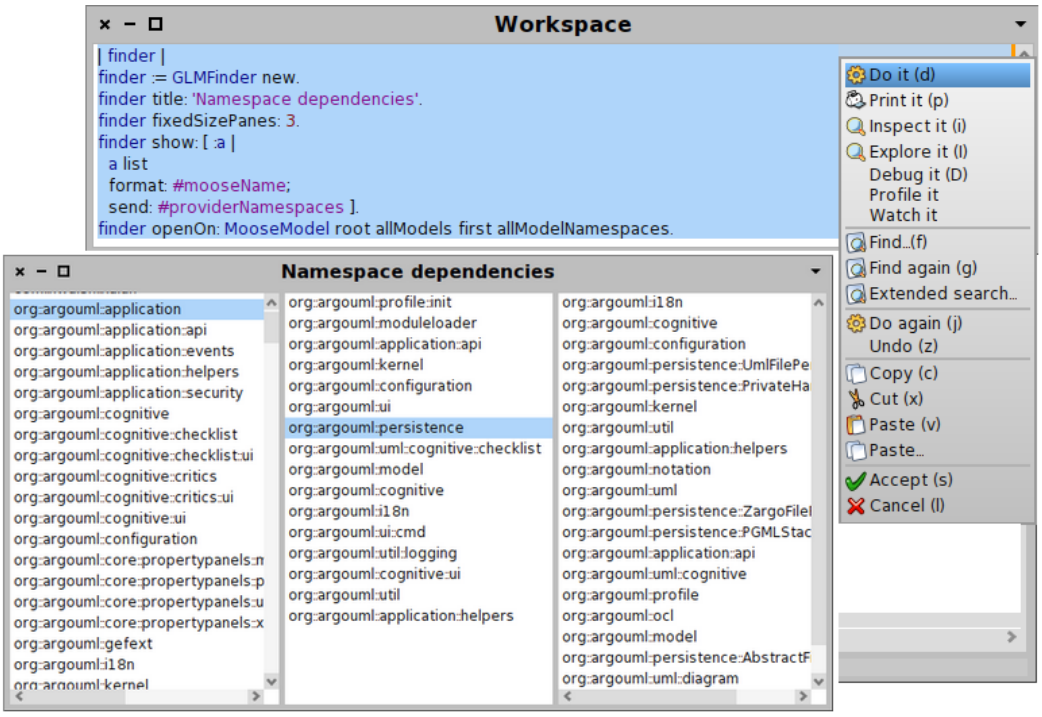
[Mondrian Easel opened on all model classes from the output case study]



- Finder:

Finder is an implicit browser and it takes its name from the browser used to navigate through the file system of Mac OS X.

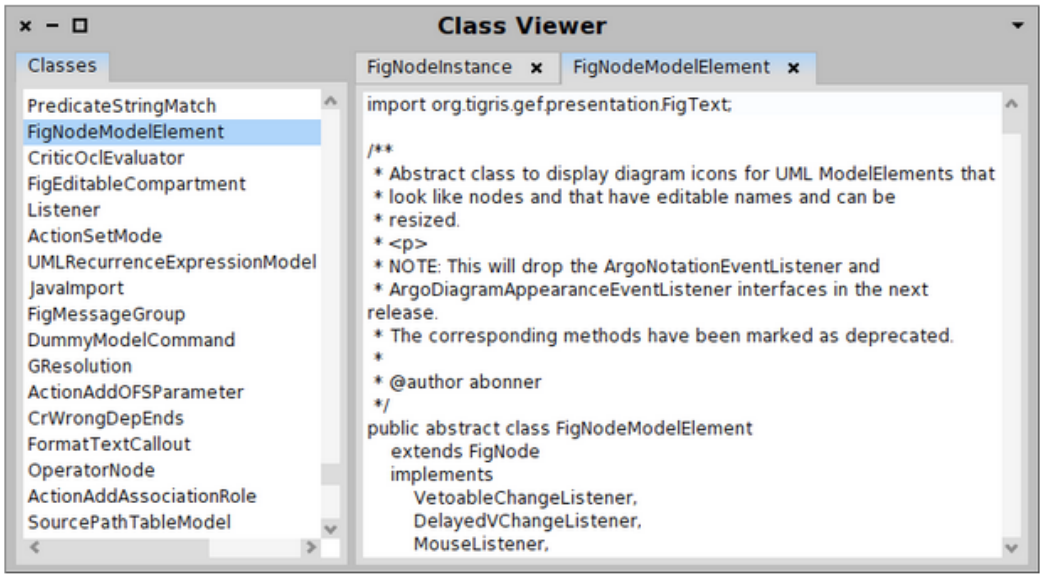
To explore the provider Namespaces for each namespace from output model using a Finder. In this example, I will use a list for each pane showing the current list of namespaces. Selecting a namespace spawns the provider namespaces to the right.



[Finder example showing the dependencies of the namespaces from ArgoUML]

- Accumulator:

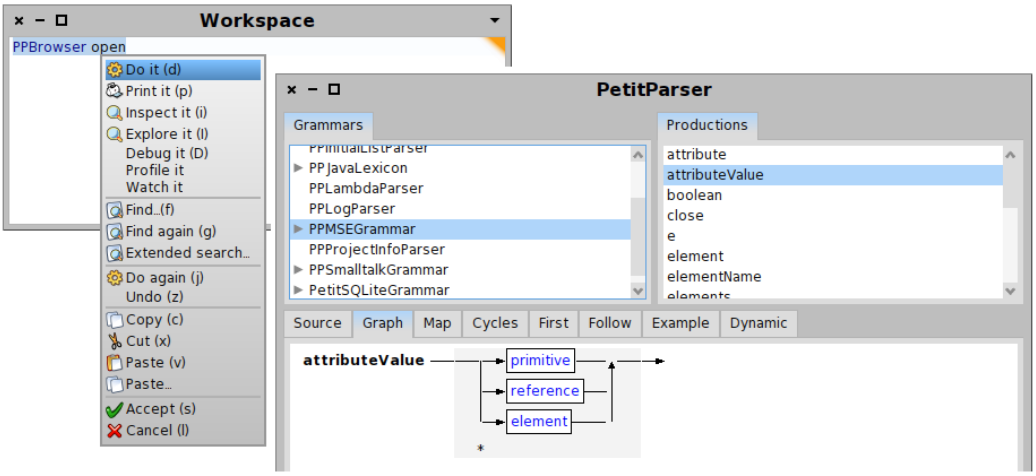
Accumulator is an implicit type of browser whose only behavior is to accumulate panes.



[Accumulator example simulating a viewer for the classes from ArgUML case study]

- Dedicated user interface:

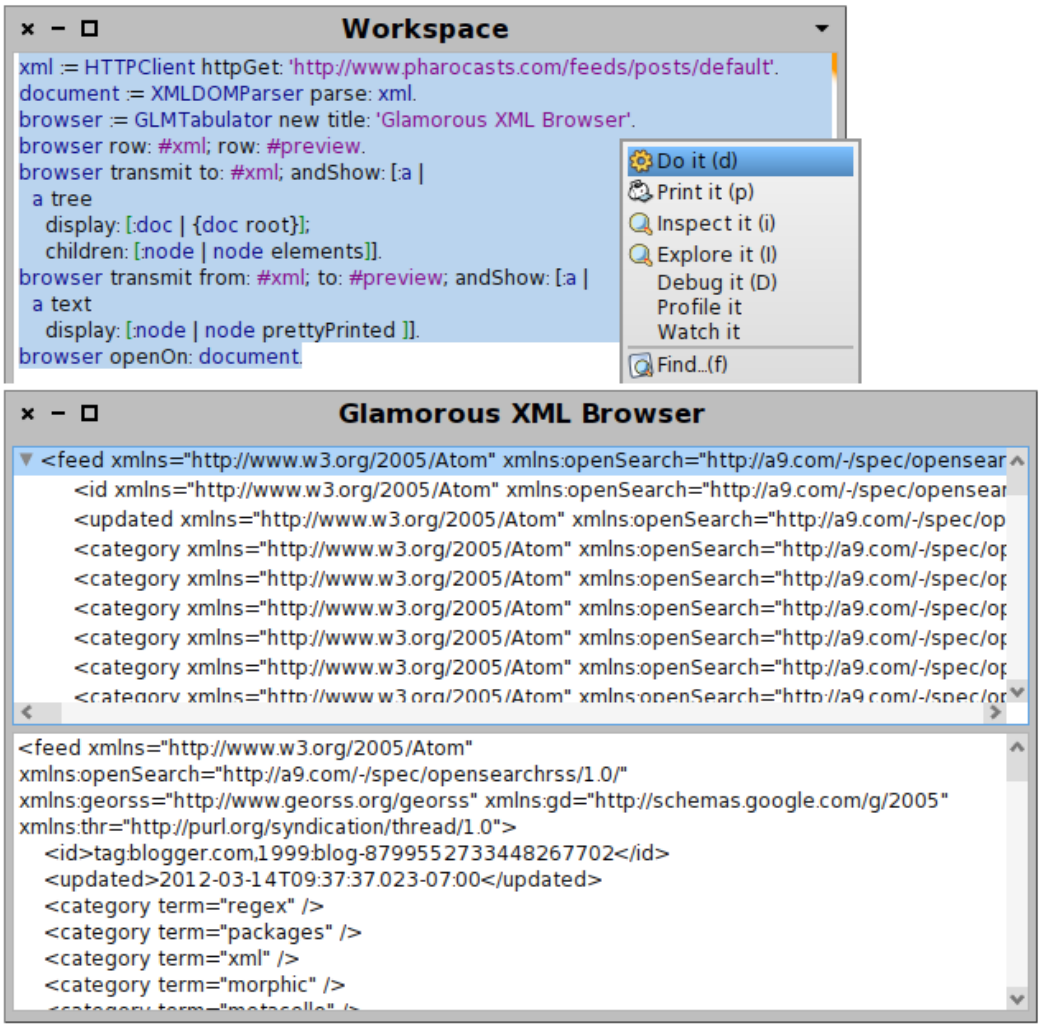
If you want to visualize, edit, or debug the parser, you can use the built-in editor by:



- Glamorous Toolkit (GT):

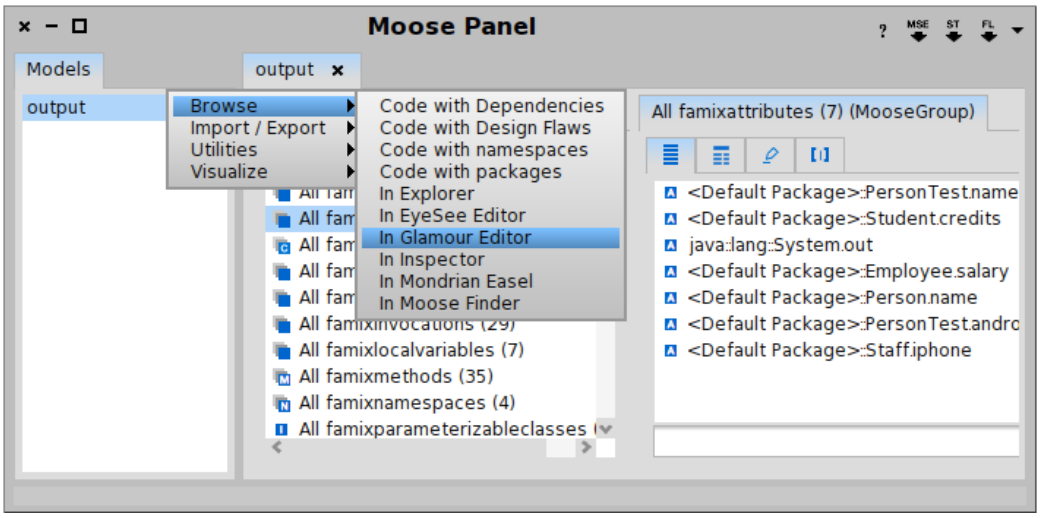
The Glamorous Toolkit is a set of tools for Smalltalk development. It is built on top of Glamour. Glamour is an engine for scripting browsers.

- Building a simple xml browser with Glamour [2]:

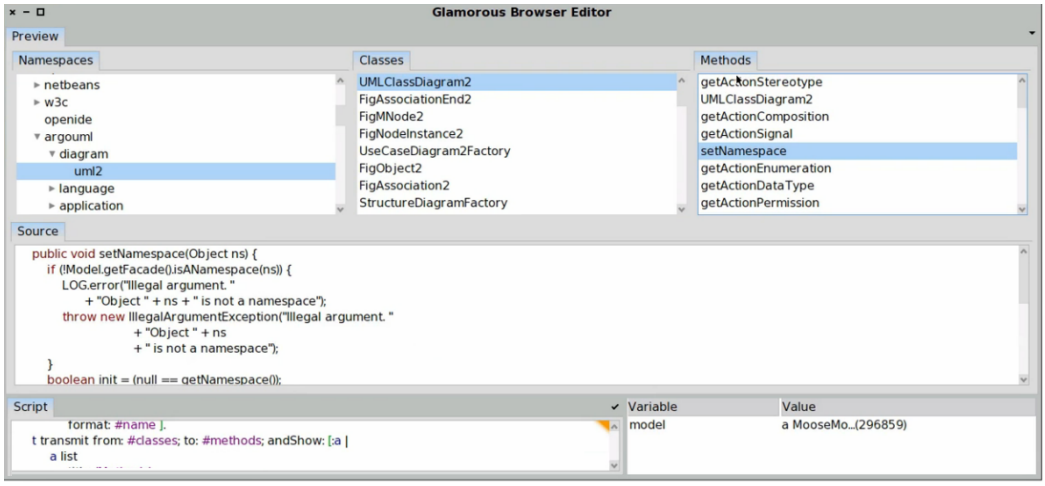


[Building a simple xml browser with Glamour]

- Building a simple code browser:




```
composer tabulator with:[:t
	t
	row:[:r |r column:#namaespaces;column:#clasess;column:#methods];
	row:#details.
	t transmit to:#namesapces; andShow:[:a |
		a tree
		title:'Namespaces';
		display:[:mooseMode|mooseMode allNamespaces select:#isRoot];
		children:#childScopes;
		format:#name].
	t transmit from:#namespaces;to:#clasess;andShow:[:a|
		a list
		title:'Classes';
		display:#classes;
		format:#name].
	t transmit from:#clasess;to:#methods; andShow:[:a|
		a list
		title:'Methods';
		display:#methods;
		format:#name].
	t transmit from:#methods;to:#details;andShow:[:a
		a text
		title:'Source';
		display:#FormattedSourceText].
	].
composer startOn: model
```


Glamorous Browser Editor
Preview
Namespaces
netbeans
w3c
openide
argouml
diagram
uml2
language
application
Classes
UMLClassDiagram2
FigAssociationEnd2
FigMNode2
FigNodeInstance2
UseCaseDiagram2Factory
FigObject2
FigAssociation2
StructureDiagramFactory
Methods
getActionStereotype
UMLClassDiagram2
getActionComposition
getActionSignal
setNamespace
getActionEnumeration
getActionDataType
getActionPermission
Source
public void setNamespace(Object ns) {
if (!Model.getFacade().isANamespace(ns)) {
LOG.error("Illegal argument. "
+ "Object " + ns + " is not a namespace");
throw new IllegalArgumentException("Illegal argument. "
+ "Object " + ns
+ " is not a namespace");
}
boolean init = (null == getNamespace());
Script
format: #name].
t transmit from: #classes; to: #methods; andShow: [:a |
a list
Variable
Value
model
a MooseMo...(296859)

Full Example:

1. Download Eclipse IDE for Java Developers from http://www.eclipse.org/downloads/
2. Create the following Java Project using Eclipse IDE:

A. Main Class:

```
public class Main {
public static void main(String []args){

PersonTest person1 =new PersonTest();
person1.setName("Ra'Fat" + " AL-Msie'DeeN");
person1.setAndroid(" Hello Student");

Person person2 =new Person("Rami AL-Msie'DeeN");

Staff staff1 =new Staff();
staff1.setIphone("Hello Staff");

Student student1=new Student("Fadi AL-Msie'DeeN ");
Student student2=new Student("Saja AL-Msie'DeeN");
student1.setCredits(2000);
student2.setCredits(5000);

Employee employee1=new Employee("Ahmad AL-Msie'DeeN");
employee1.setSalary(10000);

System.out.println(person1);
System.out.println(person2);

System.out.println(staff1);

System.out.println(student1);
System.out.println(student2);

System.out.println(employee1);}}
```

B. Staff Class:

```
public class Staff {

private String iphone;

public String getIphone() {return iphone;}

public void setIphone(String iphone) {this.iphone = iphone;}

public String toString() {return iphone;}

public static void main(String[] args) {}}
```

C. PersonTest Class:

```
public class PersonTest {
private String name;
private String android;
public String getName() {return name;}
public void setName(String name) {this.name = name;}
public String getAndroid() {return android;}
public void setAndroid(String android) {this.android = android;}
public String toString() {return name + android;}
public static void main(String[] args) {}}
```

D. Person Class:

```
public class Person {
private String name;
public Person(String name){this.name=name;}
public String getName() {return name;}
public void setName(String name) {this.name = name; }
public String toString() {return name;}
public static void main(String[] args) {}}
```

E. Student Class:

```
public class Student extends Person {
public Student(String Name) {super(Name);}
//Person(String Name) Constructor
private int credits;
public int getCredits() {return credits;}
public void setCredits(int credits) {this.credits = credits;}
public String toString() {return getName()+" Credits:" + credits ;}
public static void main(String[] args) {}}
```

F. Employee Class:

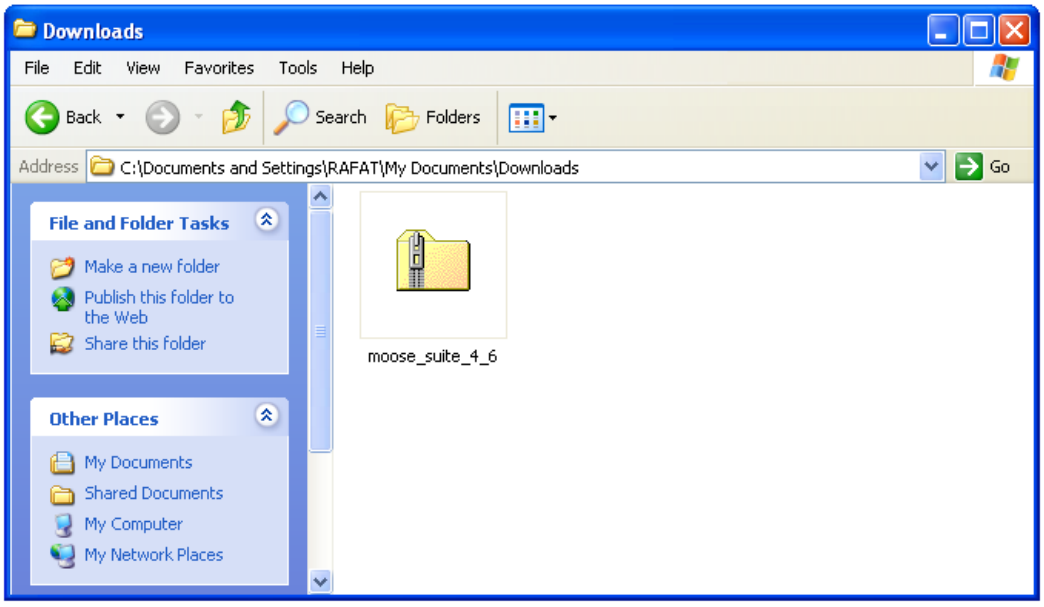
```
public class Employee extends Person {
public Employee(String name) {super(name);}
// Person(String Name) Constructor
private double salary;
public double getSalary() {return salary;}
public void setSalary(double salary) {this.salary = salary;}
public String toString() {return getName() + " Salary:" + salary;}
public static void main(String[] args) {}}
```

G. Overloading Class:

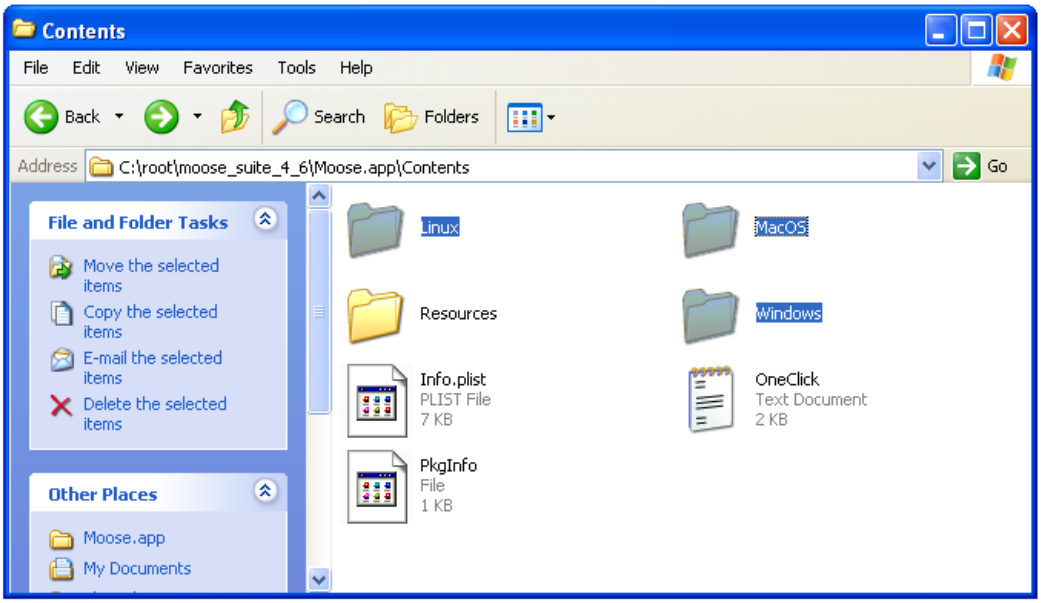
```
public class Overloading {
protected void Sum(int a){int sum1=1+a; System.out.println(sum1);}
private void Sum(int a,int b){int sum1=a+b;
System.out.println(sum1);}
public void  Sum(int a,int b,int c){int sum1=a+b+c;
System.out.println(sum1);}
public static void main(String[] args){
Overloading roro=new Overloading();
roro.Sum(12);
roro.Sum(12, 12);
roro.Sum(21,21, 21);
}}
```

3. Download and Extract Moose Suite 4.6 Folder:

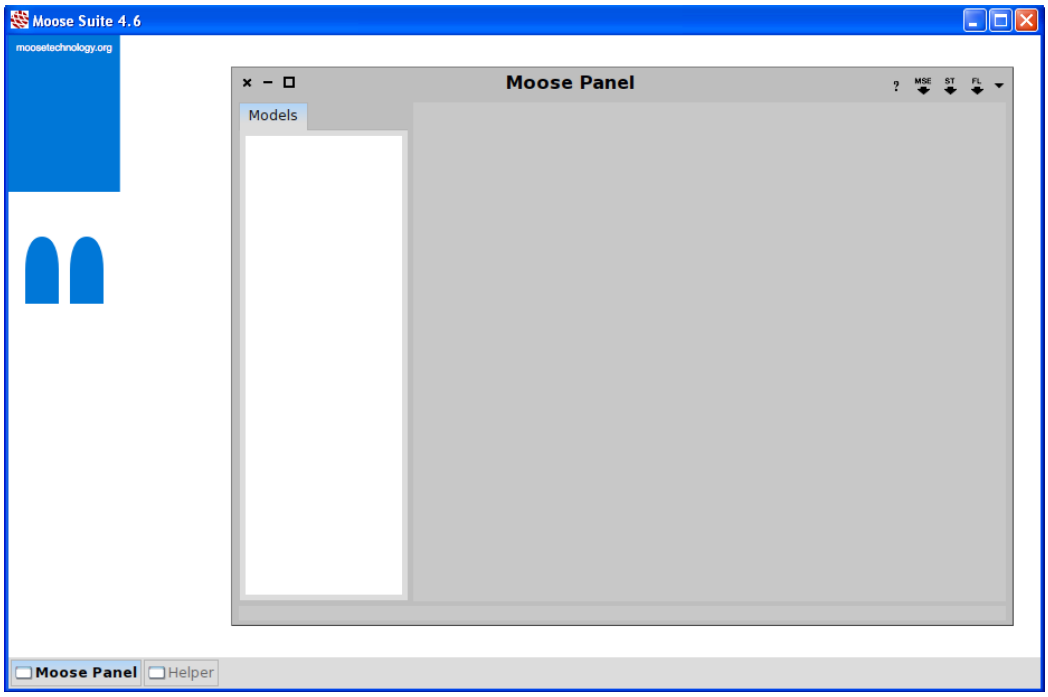
   http://www.moosetechnology.org/download



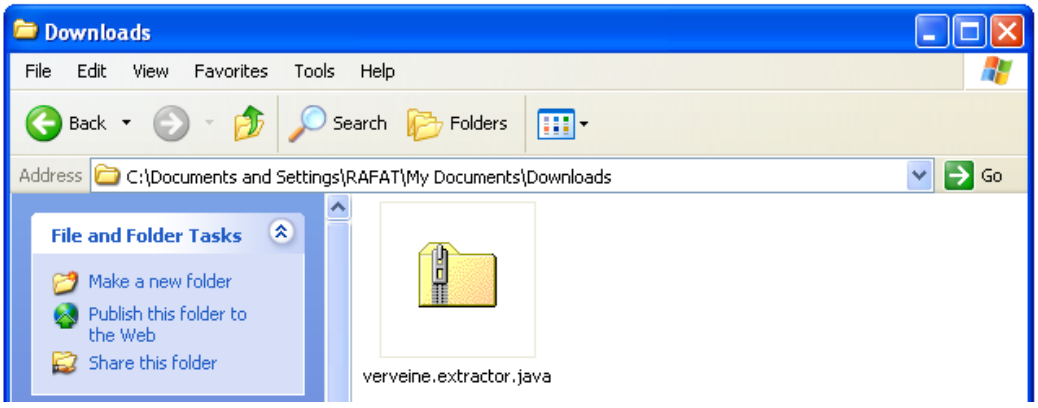
4. Extract the Folder:



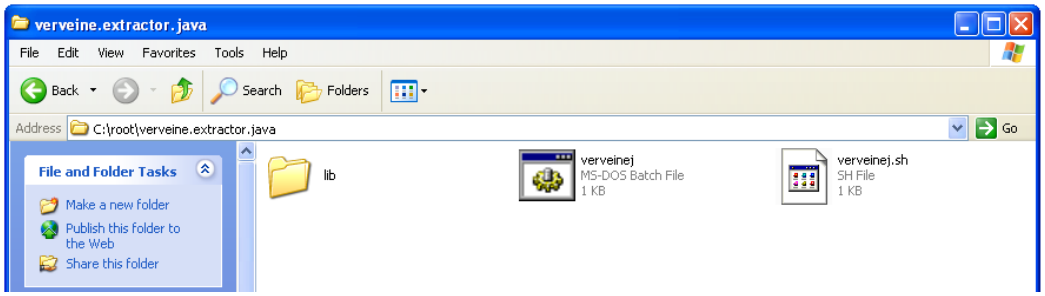
- Launch the platform specific executable:
  - Mac: launch the app file
  - Linux: launch .app/moose.sh
  - Windows: launch .app/moose.lnk
  - Launching the executable spawns a window as in the picture below:



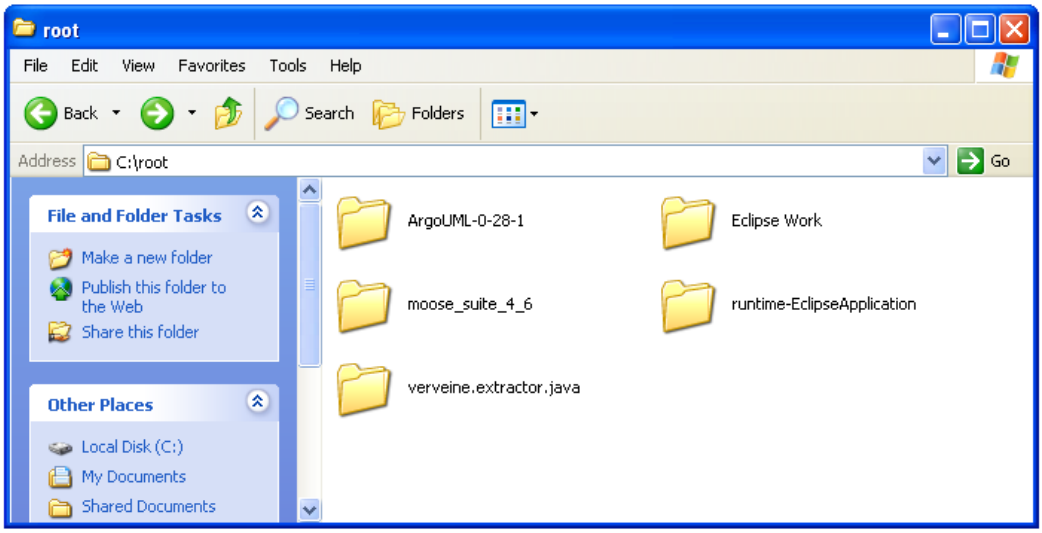
5. Download and Extract the latest verveine.extractor.java folder: http://www.moosetechnology.org/tools/verveinej



6. Extract the Folder:



7. Create the root folder of your Java project [the one that includes all sources]:



8. Run VerveineJ and Produced the MSE File:

A. For Mach Operating System:

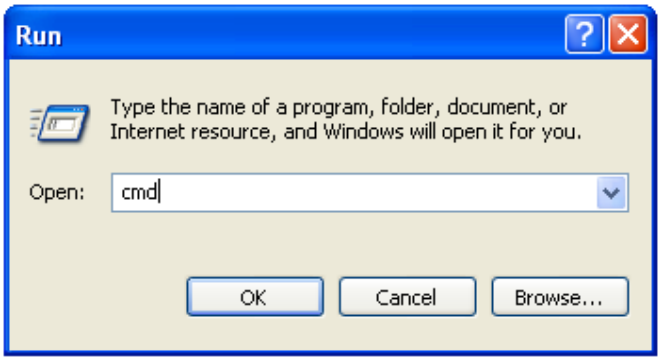
- Go to the root folder of your Java project (the one that includes all sources).
- Run the script to produce the MSE file: verveinej.sh.
- Import the output.MSE file into Moose.
- The complete command line options for running VerveineJ take the following form:

verveinej.sh [<jvm-options> --] [<verveine-options>] <java-source>

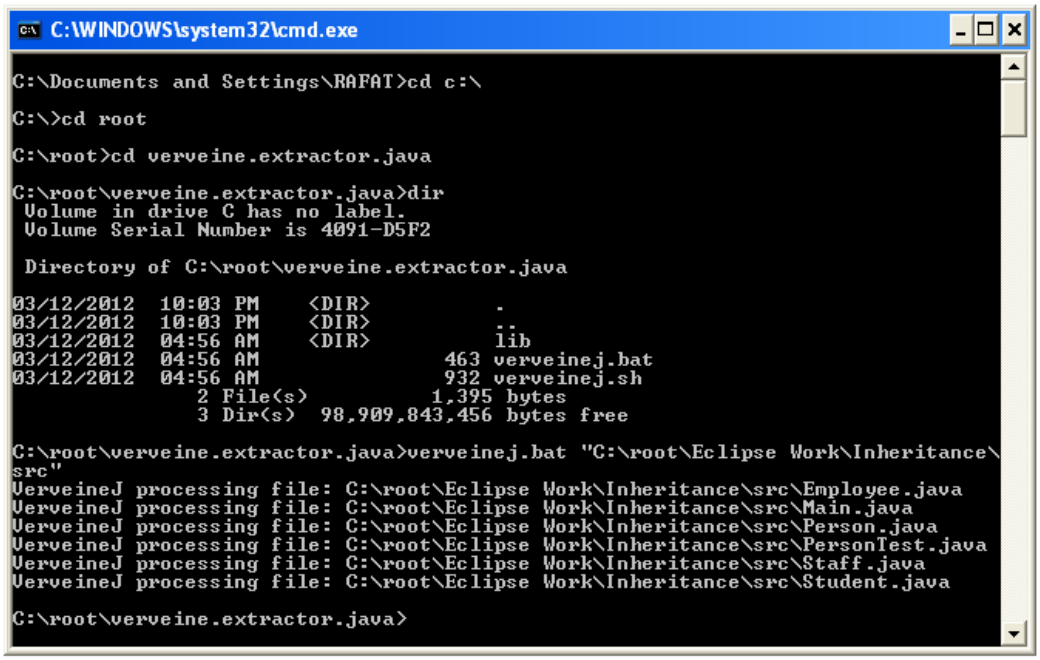
- Note: The exported MSE model only contains the pointer to the source code, but not the actual source. Thus, when loading into Moose, you need to tell Moose where the root folder is.

B. For Windows XP Operating System:

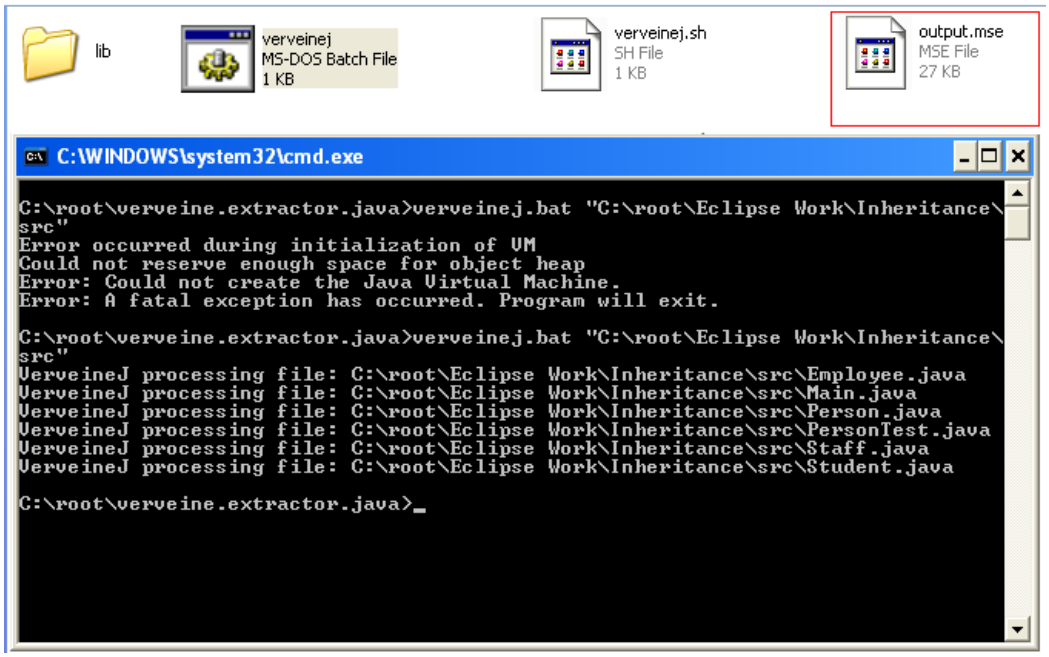
- Run the Command Line Interface: start >> run.



- Access the root Folder from command line, after that access the verveine.extractor.java folder using command line.
- Run the verveinej.bat and access the root folder of your Java project; the one that includes all sources, after that the output.mse file will produce.
- Import the output.MSE file into Moose.
- The complete command line options for running VerveineJ take the following form:



- The output.mse File:



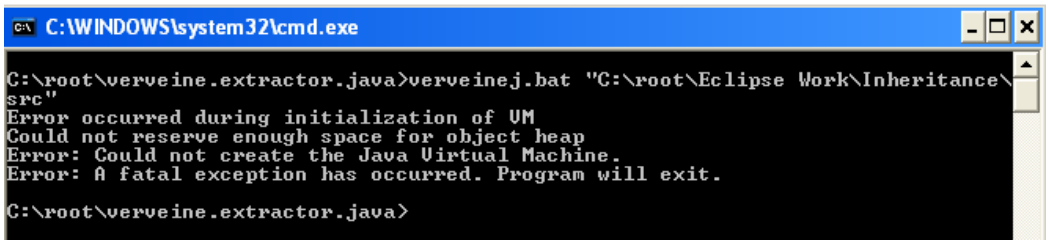
- VerveineJ with More Memory: A useful option is the one that provides VerveineJ with More Memory:

```
rem JVM options e.g. -Xmx2500m to augment maximum memory size of the vm to 2.5Go.
set JOPT="-Xmx2500m"
```

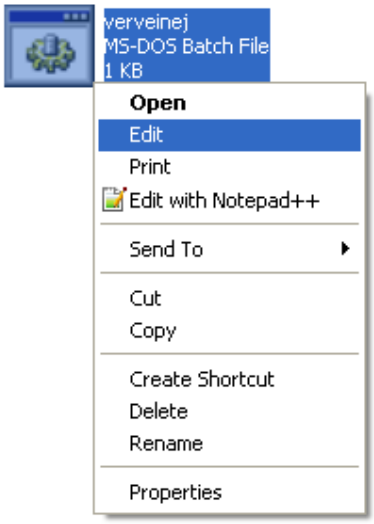
- This tells the extractor that we need the Java virtual machine to allocate 2GB, and to parse sources from the current directory.

- VerveineJ with less Memory:

- Command Line Interface Problem with VerveineJ Memory:



- To Deal with this problem, go to the verveinej.bat file, open it and edit it using Notepad ++.

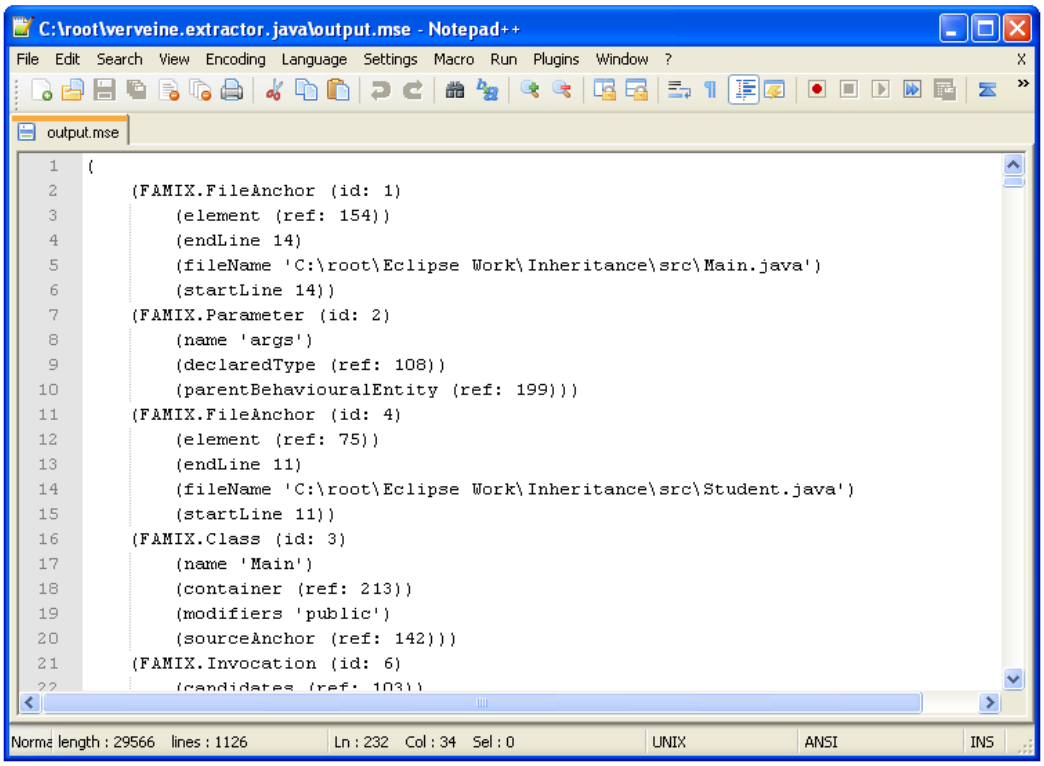


verveinej.bat

```
1  @echo off
2  setlocal ENABLEDELAYEDEXPANSION
3
4  rem Directory for verveine source
5  set ROOT=%~dp0%
6  set BASELIB=%ROOT%lib
7
8  rem JVM options e.g. -Xmx2500m to augment maximum memory size of the vm to 2.5Go.
9  set JOPT="-Xmx1000m"
10 rem Verveine option
11 rem set VOPT="."
12
13 FOR /R %BASELIB% %%G IN (*.jar) DO set LOCALCLASSPATH=%%G;!LOCALCLASSPATH!
14 set CLASSPATH=%CLASSPATH%;%LOCALCLASSPATH%
15
16 java %JOPT% fr.inria.verveine.extractor.java.VerveineJParser %1 %2 %3 %4 %5 %6 %7 %8 %9
```

- Save the verveinej.bat file after edit it.

9. Open output.mse File using Notepad ++.



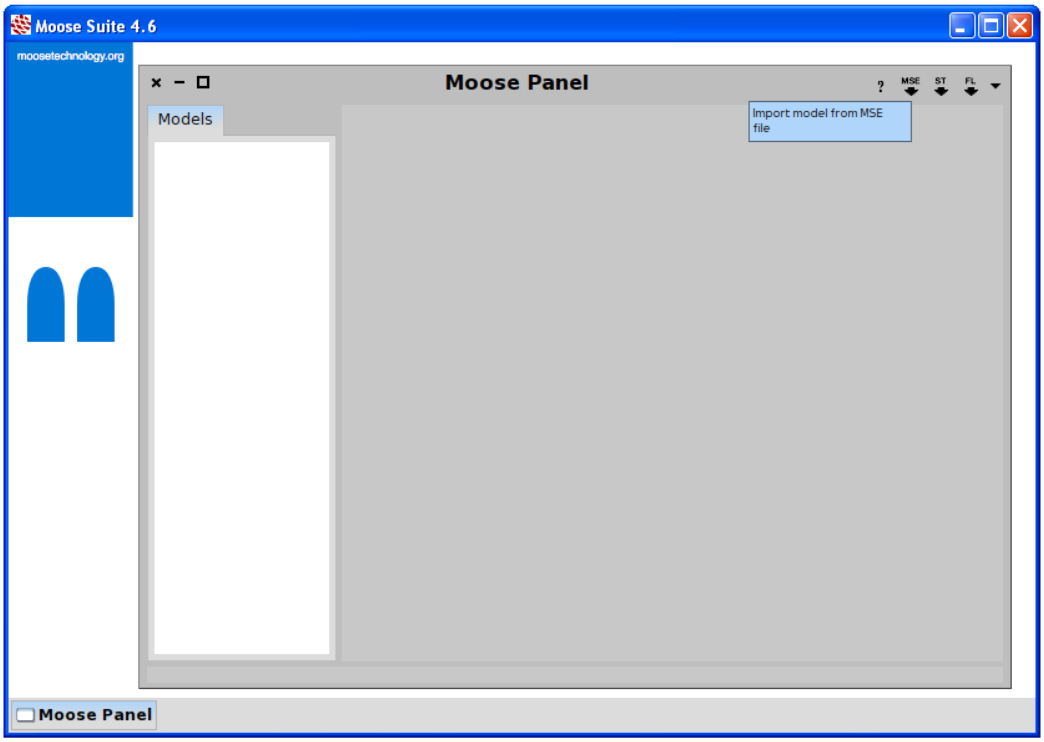
10. Importing Java and C/C++ using the inFamix external parser:

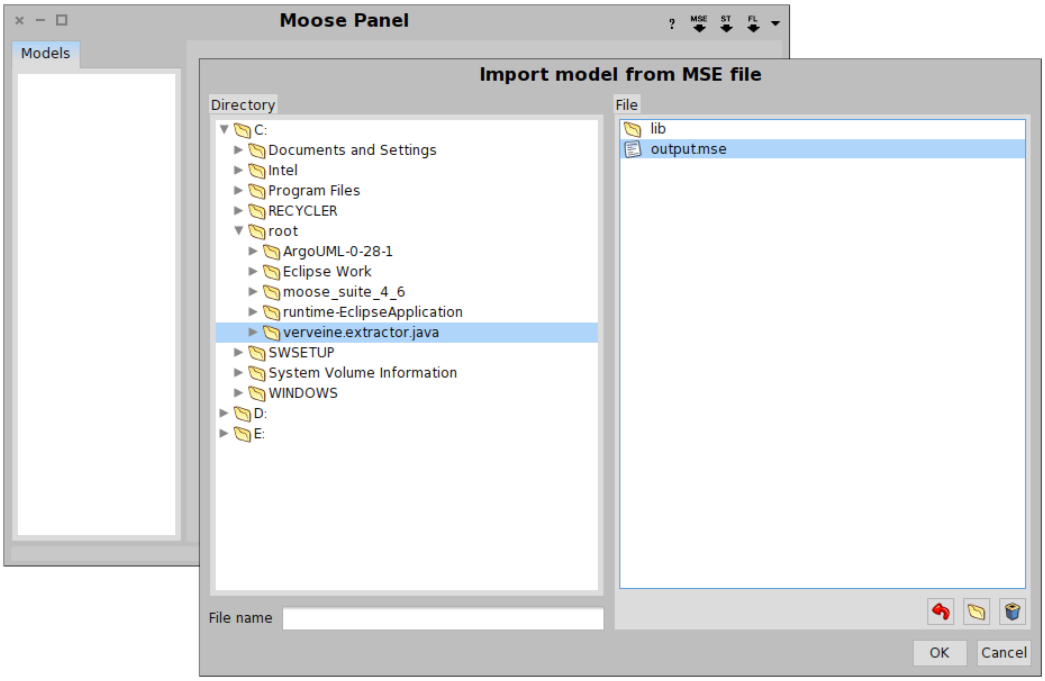
- inFamix is a parser for Java and C/C++ built by: http://www.intooitus.com/products/infamix.
- The parser is based on the scalable inFusion platform. The parser is available as a free command line tool and it exports MSE.
- Download and Extract the latest inFamix_win32 Folder.
- Launch it from the command line using the appropriate syntax, and exports MSE File.

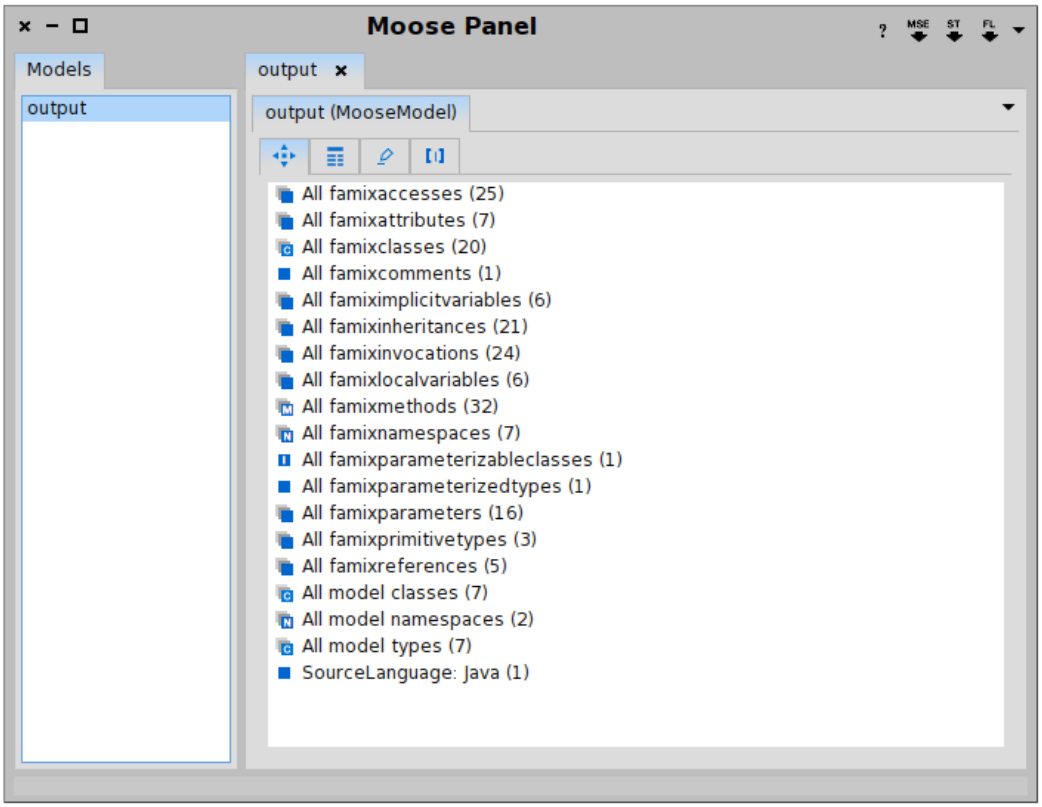
11. Importing and exporting with MSE:

- The preferred way to load a model in Moose is via an MSE file.
- Press the “Import Model from MSE” button in the Moose to load an MSE file.
- This creates a model, populates it with the entities from the file and adds the model to the repository.
- But what exactly is MSE? MSE is the default file format supported by Moose. It is a generic file format and can describe any model.
- MSE similar to XML, the main difference being that instead of using verbose tags, it makes use of parentheses to denote the beginning and ending of an element.

```
( (FAMIX.Namespace (id: 1)
    (name 'aNamespace'))
```




Moose Panel
Models
Import model from MSE file
Directory
C:
Documents and Settings
Intel
Program Files
RECYCLER
root
ArgoUML-0-28-1
Eclipse Work
moose_suite_4_6
runtime-EclipseApplication
verveine.extractor.java
SWSETUP
System Volume Information
WINDOWS
D:
E:
File
lib
output.mse
File name
OK
Cancel


Moose Panel
Models
output
output (MooseModel)
All famixaccesses (25)
All famixattributes (7)
All famixclasses (20)
All famixcomments (1)
All famiximplicitvariables (6)
All famixinheritances (21)
All famixinvocations (24)
All famixlocalvariables (6)
All famixmethods (32)
All famixnamespaces (7)
All famixparameterizableclasses (1)
All famixparameterizedtypes (1)
All famixparameters (16)
All famixprimitivetypes (3)
All famixreferences (5)
All model classes (7)
All model namespaces (2)
All model types (7)
SourceLanguage: Java (1)

References:

[1] "Moose technology: Home," 2012. From http://www.moosetechnology.org/

[2] http://www.tudorgirba.com/blog/simple-xml-browser-with-glamour.