glamour

Scripting Browsers[*]

Philipp Bunge, Tudor Gîrba, Lukas Renggli,
Jorge Ressia, David Röthlisberger

Software Composition Group, University of Bern, Switzerland

1 Glamour in a Nutshell

Browsers are crucial to make software models accessible. Problem domains often require multiple views to access, interpret and edit the underlying elements. However, browsers are expensive to create and burdensome to maintain.

Glamour is a framework dedicated to building browsers. It uses a components and connectors architecture and it comes with an embedded domain specific language that allows the user to build dedicated browsers quickly. It accommodates any kind of domain models via on-the-fly transformations and it enforces a strict and explicit separation between the presentation of the data and the navigation flow between different entities.

Glamour is platform independent and can be used with various renders. For details, we refer the reader to the Masters thesis of the first author [Bun09].

[*]Submission for the ESUG 2009 Innovation Technology Awards. Glamour was awarded the 3rd prize.

2 Glamour in Action

As a practical example of Glamour we show how to implement a simple Smalltalk code browser (Figure 1) with less than 20 lines of code. A Glamour script consists of two parts. The first part defines the overall location of the different panes in the window. The second part defines how the panes should present the entities and what the navigation between these panes is.

First we specify that the browser has a table layout. The upper part contains the class tree and the method for each class and the lower part displays the source code. The following code defines two rows. In the first row we create two columns: one for the classes and one for the selectors. The second row is for displaying the source code.

```
 1 browser := GLMTableLayoutBrowser new.
 2 browser row: [ :row | row column: #classes; column: #selectors ].
 3 browser row: #source.
```

Next we specify the presentation to be shown in each pane of the browser. For the class pane we use a tree to display the class hierarchy.

```
 4 browser showOn: #classes; using: [
 5    browser tree
 6       children: [ :class | class subclasses ] ].
```

For the selector pane we use a list, that displays the selectors of the currently selected class. This dependency is specified in the `from:`-clause of the following code:

```
 7 browser showOn: #selectors; from: #classes; using: [
 8    browser list
 9       display: [ :class | class selectors ] ].
```

Finally, the source code pane depends on both, the currently selected class and the selector. We also define two different text views. The first one shows the source code, given a class and a selector. The second one shows the class comment for a given class. In both cases we define conditions (lines 13 and 17) when the presentations should be active. If both conditions are satisfied a tab panel is created.

```
10 browser showOn: #source; from: #classes; from: #selectors; using: [
11    browser text
12       title: 'Source';
13       when: [ :class :selector | class notNil and: [ selector notNil ] ];
14       display: [ :class :selector | class sourceCodeAt: selector ].
15    browser text
16       title: 'Comment';
17       when: [ :class :selector | class notNil ];
18       display: [ :class :selector | class comment ] ].
```

To open the browser we evaluate the following code. Depending on the chosen renderer a Morphic (see Figure 1(a)), Seaside (see Figure 1(b)), Widgetry (see Figure 1(c)) or Adobe Air (see Figure 1(d)) user interface is opened.

```
browser openOn: Object.
```

Note that the browser can be composed to new and more complicated browsers. Although it is possible to attach context menus and actions to the different panes. Furthermore other presentations are available, such as Mondrian visualizations [MG06] and Magritte forms [RDK07].

3 Availability

Glamour is developed in Pharo and VisualWorks. The Glamour model is independent of the resulting user interface. Currently, Polymorphic and Seaside are supported in Pharo, and Wrapper and Adobe Air in VisualWorks.

- **Pharo.** To load into a recent Pharo image, load the package `GlamourLoader` from SqueakSource at `http://www.squeaksource.com/Glamour`.
- **VisualWorks.** To load into VisualWorks, load the bundle `Glamour` from Postgres StORE at `db.iam.unibe.ch:5432_scgStore`, use `storeguest` as both username and password.

More details about Glamour can be found at: `http://moose.unibe.ch/tools/glamour`

4 Keywords

Browser, User Interface, Scripting, Pharo, VisualWorks, Seaside, Adobe Air

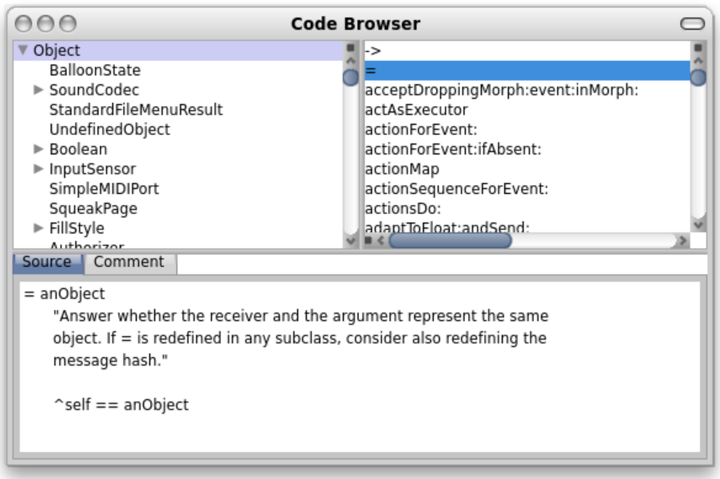
5 License

Glamour is open-source Software distributed under the MIT license, that grants unrestricted copy, redistribution, usage and embedding in both free and proprietary software.

References

[Bun09] Philipp Bunge. Scripting browsers with Glamour. Master's thesis, University of Bern, April 2009.

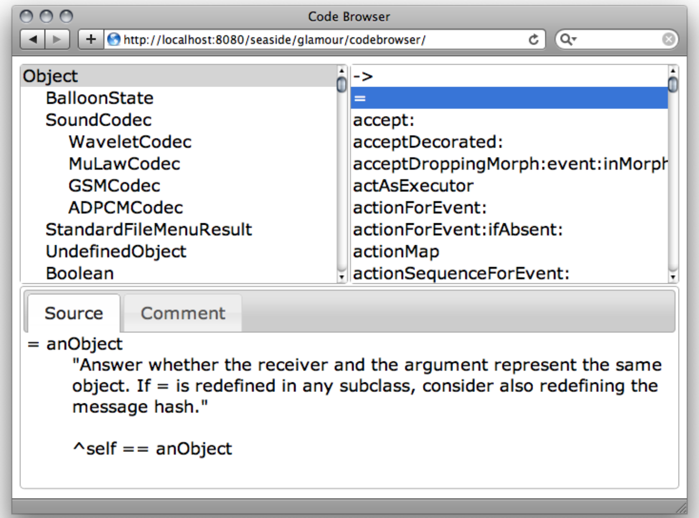
[MG06] Michael Meyer and Tudor Gîrba. Mondrian: Scripting visualizations. European Smalltalk User Group 2006 Technology Innovation Awards, August 2006. It received the 2nd prize.



```
= anObject
	"Answer whether the receiver and the argument represent the same
	object. If = is redefined in any subclass, consider also redefining the
	message hash."

	^self == anObject
```

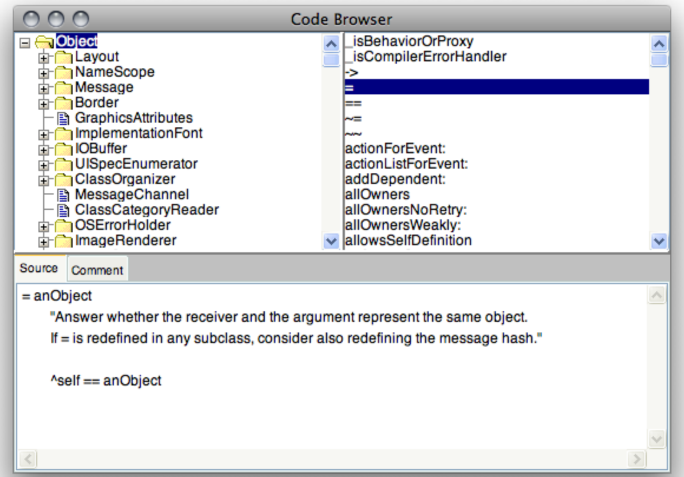
(a) Morphic



```
= anObject
	"Answer whether the receiver and the argument represent the same
	object. If = is redefined in any subclass, consider also redefining the
	message hash."

	^self == anObject
```

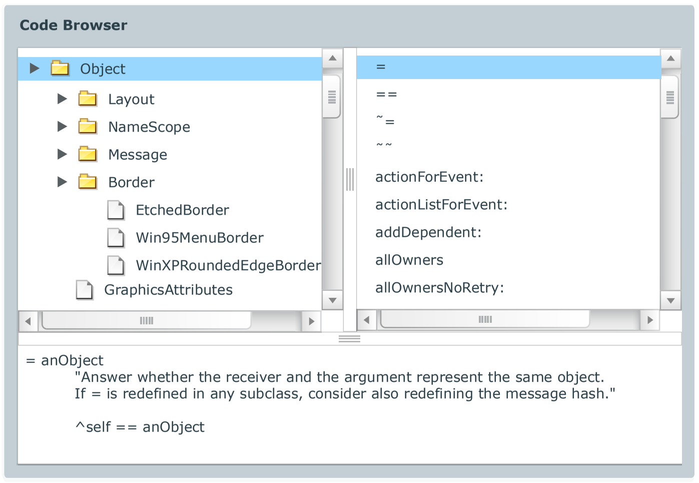
(b) Seaside



```
= anObject
	"Answer whether the receiver and the argument represent the same object.
	If = is redefined in any subclass, consider also redefining the message hash."

	^self == anObject
```

(c) Widgetry



```
= anObject
	"Answer whether the receiver and the argument represent the same object.
	If = is redefined in any subclass, consider also redefining the message hash."

	^self == anObject
```

(d) Adobe Air

Figure 1: The same Glamour code browser displayed using four different output frameworks.

[RDK07] Lukas Renggli, Stéphane Ducasse, and Adrian Kuhn. Magritte — a meta-driven approach to empower developers and end users. In Gregor Engels, Bill Opdyke, Douglas C. Schmidt, and Frank Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *LNCS*, pages 106–120. Springer, September 2007.