The Roassal Visualization Engine

The following tutorial is designed for students to get used with basic concepts of Roassal. At the end of the tutorial, the student should be able to know how to use basic features of the Roassal engine to visualize software metrics and relationships within a system by using visual attributes such as shapes, a layout, color, size, as well as simple interactions. Students should also be able to know where to find more information to resolve doubts and to learn more about the engine. The tutorial is split in three parts: 1) Hand’s on session, 2) Examples of Roassal’s builders and 3) Pointers to find more information. Let’s start.

How to install

I. Download the latest version of Moose
II. If you want to use the Pharo image you can follow the instruction in the Roassal website

a) Hands-on

1. Create a view, elements, and add them the elements to the view

```plaintext
classes := RTObject withAllSubclasses.
view := RTView new.
elements := RTEllipse new
elementsOn: classes.
view addAll: elements.
view
```

2. Add interactions, enable elements to be dragged.

```plaintext
| classes := RTObject withAllSubclasses. |
| view := RTView new. |
| elements := RTEllipse new |
| elementsOn: classes. |
| view addAll: elements. |
| view
```

```plaintext
| classes := RTObject withAllSubclasses. |
| view := RTView new. |
| elements := RTEllipse new |
| elementsOn: classes. |
| elements @ RTDraggable. |
| elements @ RTPopup text:[e asString]]. |
| view @ RTDraggableView. |
| view
```

3. Use a grid layout to avoid overlapping elements
4. Assign an arbitrary color to the elements in the view

5. Use color to identify elements that represent the classes of the RTLayout hierarchy

6. Encode metrics in the size of the boxes
7. Add edges to connect class hierarchies

8. Change the layout used in the visualization
9. Use circles, and encode 1) NOM in the size, and 2) NLOC in the color. Also push the edges to the back
10. Add Bezier edges to connect dependent classes

11. Use a normalizer to adjust the size of the elements. Push Bezier edges to the front.
2. Builders

A builder is a domain-specific language for agile visualization in a particular. Users of the Roassal visualization engine can define custom builders by subclassing the `RTBuilder` class. The custom builder has to define public methods for end-users to specify how the objects will be visualized, the layouts that can be applied, and other attributes specific to the domain. The custom builder has to override the `renderIn:` method where it specifies the details of how the visualization is going to be built. The following examples (amongst others) of domain-specific builders are found in Roassal Examples (within the Moose/Pharo image).

1. UML
2. RTCharterBuilder
3. RTPieBuilder
4. RTSpectrograph
5. RTSunburstBuilder
6. RTGraphBuilder
7. RTNameCloud
8. RTMondrianViewBuilder
9. RTMapLocationBuilder

3. Resources

I. http://agilevisualization.com
II. Deep into Pharo book
III. Mailing lists Pharo / Moose

Dr. Leonel Merino
Postdoctoral Researcher
Visualization Research Center (VISUS)
University of Stuttgart
leonel.merino@visus.uni-stuttgart.de

14-nov-2018