Roassal Tutorial (visualization engine)

The following tutorial is meant for the student to get used with basic concepts of Roassal. At the end of the tutorial, the student should be able to know how to use Roassal to visualize object’s metrics and relationships by using shapes, a layout, color, size and interactions. He 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.

1. Installation
	1. *Download the latest version of Moose*
	2. *If you want to use the Pharo image you can follow the instruction in the Roassal website*
2. Hands-on (***blue*** *🡪 added code;* ***red*** *🡪 removed code)*
	1. View, Shape

*Goal: Show a collection of elements with a grid layout.*

* + 1. Create a RTView object
		2. Depict RTObject subclasses using a RTEllipse shape
		3. Apply a RTGridLayout

view := RTView new.

shape := RTEllipse new.

els := shape elementsOn: RTObject withAllSubclasses.

view addAll: els.

RTGridLayout new gapSize:15; on: els.

view open

* 1. Shape, color and size

*Goal: Specify a different shape, color and size for elements.*

* + 1. Change the shape to RTBox
		2. Set height in 20 and width in 10
		3. Set a random color for all/each nodes

view := RTView new.

shape := ~~RTEllipse~~ **RTBox** new **height:20; width:10; color:[:e|Color random]**.

els := shape elementsOn: RTObject withAllSubclasses.

view addAll: els.

RTGridLayout new gapSize:15; on: els.

view open

* 1. Edges

*Goal: Show the class hierarchy through edges with a tree layout.*

* + 1. Add edges from each class element to its subclasses
		2. Change the layout to a RTTreeLayout

view := RTView new.

shape := RTBox new height:20; width:10; color:[:e|Color random].

els := shape elementsOn: RTObject withAllSubclasses.

view addAll: els.

**RTEdge buildEdgesFromObjects: RTObject withAllSubclasses from:#yourself toAll:#subclasses using: RTLine inView: view.**

~~RTGridLayout new gapSize:15;~~ **RTTreeLayout** on: els.

els @ RTDraggable.

els @ RTPopup.

els @ (RTHighlightable new highlightedColor: Color red).

els @ (RTMenuActivable new action:#inspect).

view @ RTDraggableView.

view open

* 1. Interactions

*Goal: Try interactions for elements and for the view.*

* + 1. Set elements RTDraggable
		2. Add Popup to elements
		3. Set elements RTHighlightable (with red color)
		4. Add a RTMenuActivable to inspect elements
		5. Set elements RTLabelled
		6. Specify the layout’s gapSize as 15
		7. Set view RTDraggableView

view := RTView new.

shape := RTBox new height:20; width:10; color:[:e|Color random].

els := shape elementsOn: RTObject withAllSubclasses.

view addAll: els.

RTGridLayout new gapSize:15; on: els.

**els @ RTDraggable.**

**els @ RTPopup.**

**els @ (RTHighlightable new highlightedColor: Color red).**

**els @ (RTMenuActivable new action:#inspect).**

**view @ RTDraggableView.**

view open

* 1. Metrics

*Goal: Define and map metrics to visual dimensions.*

* + 1. Define height as LOC (Number of Lines of Code)
		2. Define width as NOM (Number of Methods)
		3. Define color as NDC (Number of dependent classes)
		4. Show NOD in the Popup

view := RTView new.

shape := RTBox new **height:#numberOfLinesOfCode; width:[:e| e methods size]; color:[:e|Color grayScale256: (256 - e dependentClasses size)]**.

els := shape elementsOn: RTObject withAllSubclasses.

view addAll: els.

RTEdge buildEdgesFromObjects: RTObject withAllSubclasses from:#yourself toAll:#subclasses using: RTLine inView: view.

RTTreeLayout on: els.

els @ RTDraggable.

els @ **(**RTPopup **new textElement:[:e| e model dependentClasses size])**.

els @ (RTHighlightable new highlightedColor: Color red).

els @ (RTMenuActivable new action:#inspect).

view @ RTDraggableView.

view open

* 1. Normalization

*Goal: Normalize the color and height properties.*

* + 1. Color using NDC metric from green to red
		2. Height using LOC metric with a square root function

view := RTView new.

shape := RTBox new ~~height:#numberOfLinesOfCode;~~ width:[:e| e methods size]~~; color:[:e|Color grayScale256: (256 - e dependentClasses size)]~~.

els := shape elementsOn: RTObject withAllSubclasses.

view addAll: els.

RTEdge buildEdgesFromObjects: RTObject withAllSubclasses from:#yourself toAll:#subclasses using: RTLine inView: view.

**RTMetricNormalizer new**

 **elements: els;**

 **normalizeHeight: #numberOfLinesOfCode min:5 max:1000 using:[:e| e sqrt];**

 **normalizeColor: [:e| e dependentClasses size ] using:{Color green . Color red}.**

RTTreeLayout on: els.

els @ RTDraggable.

els @ (RTPopup new textElement:[:e| e model dependentClasses size]).

els @ (RTHighlightable new highlightedColor: Color red).

els @ (RTMenuActivable new action:#inspect).

view @ RTDraggableView.

view open

1. Builders

A builder encapsulates the logic of visualization and it maps a particular domain to visual elements. It eases the reuse of domain-specific visualizations. Developers can define custom builders by sub classing **RTBuilder**. The custom builder should define public methods for end-users to specify the objects to be visualized, layouts that can be applied and any other attribute specific to the domain. The custom builder should override the ***renderIn:*** method where it can specify how the visualization is built. The following examples (amongst others) of domain-specific builder are found in Roassal Examples.

* 1. UML
	2. RTCharterBuilder
	3. RTPieBuilder
	4. RTSpectrograph
	5. RTSunburstBuilder
	6. RTGraphBuilder
	7. RTNameCloud
	8. RTMondrianViewBuilder
	9. RTMapLocationBuilder
1. Resources
	1. [Agilevisualization.com](http://agilevisualization.com) (book in progess)
	2. Roassal examples
	3. [Deep into Pharo book](http://deepintopharo.com)
	4. Mailing lists Pharo / Moose

Leonel Merino

PhD Student SCG

merino@scg.unibe.ch

*5-nov-2014*