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 (color, size) and Layout

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

* + 1. Create a RTView object
		2. Depict ArgoUML packages by RTEllipse shapes
		3. Apply a RTGridLayout

view := RTView new.

shape := RTEllipse new size: 10; color: Color red.

els := shape elementsOn: MooseModel root first allNamespaces.

view addAll: els.

RTGridLayout on: els.

view open

* 1. Edges

*Goal: Show the package dependency by using edges with a tree layout.*

* + 1. Add edges from each package its dependents packages
		2. Change the layout to a RTCircleLayout

view := RTView new.

shape := RTBox new height: 5; width:5; color: [Color random].

els := shape elementsOn: MooseModel root first allNamespaces.

view addAll: els.

RTEdge buildEdgesFromObjects: (MooseModel root first allInvocations first:100) from:[:e| e sender belongsTo container] toAll:[:e| e candidates collect:[:each| each belongsTo container] ] using: (RTLine new color: (Color green alpha:0.1)) inView: view.

RTCircleLayout new initialRadius:200; on: els.

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
		4. Set elements RTSetAlphaEdge
		5. Set view RTDraggableView

view := RTView new.

shape := RTBox new height: 5; width:5; color: [Color random].

els := shape elementsOn: MooseModel root first allNamespaces.

view addAll: els.

RTEdge buildEdgesFromObjects: (MooseModel root first allInvocations first:100) from:[:e| e sender belongsTo container] toAll:[:e| e candidates collect:[:each| each belongsTo container] ] using: (RTLine new color: (Color green alpha:0.1)) inView: view.

RTCircleLayout new initialRadius:200; on: els.

view @ RTDraggableView.

els @ RTDraggable.

els @ RTPopup.

els @ RTHighlightable.

els @ RTSetEdgeAlpha.

view open

* 1. Nesting

*Goal: Show the dependencies of the largest package*

* + 1. Exclude the largest package from the group
		2. Add it in a new group
		3. Add the classes as nodes nested in it
		4. Modify the edges to put focus in the largest package

view := RTView new.

max := MooseModel root first allNamespaces max:[:e| e classes size].

shape := RTBox new height: 5; width:5; color: [Color random].

els := shape elementsOn: (MooseModel root first allNamespaces reject:[:e| e classes = max]).

view addAll: els.

el := (RTBox new color: Color random) elementsOn: (MooseModel root first allNamespaces select:[:e| e classes size = max]).

view add: el.

RTNest new for: el add:[:group :pkg|

 group addAll:(RTBox new elementsOn: pkg classes).

 RTGridLayout on: group.

 ].

RTEdge buildEdgesFromObjects: ((MooseModel root first allInvocations select:[:e| e sender belongsTo container classes size = max]) first:100) from:[:e| e sender belongsTo] toAll:[:e| e candidates collect:[:each| each belongsTo container] ] using: (RTLine new color: (Color green alpha:0.1)) inView: view.

RTCircleLayout new initialRadius:200; on: els.

view @ RTDraggableView.

el @ RTDraggable.

els @ RTDraggable.

els @ RTPopup.

els @ RTHighlightable.

els @ RTSetEdgeAlpha.

view open

* 1. Metrics and properties

*Goal: Map metrics and properties to visual dimensions.*

* + 1. Define size of packages as NOC (Number of Classes)
		2. Define color red in classes (with more than 100 LOC)
		3. Define color white in classes (which are interfaces)
		4. Define color blue in classes (the rest)
		5. Add a menu to packages with inspect and browse actions
		6. Normalize the size of packages

view := RTView new.

max := MooseModel root first allNamespaces max:[:e| e classes size].

shape := RTEllipse new size: [:e| e classes size]; color: [Color random].

els := shape elementsOn: (MooseModel root first allNamespaces reject:[:e| e classes = max]).

view addAll: els.

el := (RTBox new color: Color darkGray) elementsOn: (MooseModel root first allNamespaces select:[:e| e classes size = max]).

view add: el.

RTNest new for: el add:[:group :pkg|

 group addAll:(RTBox new color: [:e| e isInterface ifTrue:[ Color white] ifFalse:[e numberOfLinesOfCode > 200 ifTrue:[Color red] ifFalse:[Color blue] ] ]; elementsOn: pkg classes).

 RTGridLayout on: group.

 group @ RTHighlightable.

 group @ RTSetEdgeAlpha.

 ].

RTEdge buildEdgesFromObjects: ((MooseModel root first allInvocations select:[:e| e sender belongsTo container classes size = max]) first:200) from:[:e| e sender belongsTo] toAll:[:e| e candidates collect:[:each| each belongsTo container] ] using: (RTArrowedLine new color: (Color green alpha:0.1)) inView: view.

RTCircleLayout new initialRadius:300; on: els.

RTMetricNormalizer new

 elements: els;

 normalizeSize:[:e| e classes size] min: 10 max: 50.

view @ RTDraggableView.

el @ RTDraggable.

els @ RTDraggable.

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

els @ RTHighlightable.

els @ (RTMenuActivable new action: #inspect; action:#browse).

els @ RTSetEdgeAlpha.

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*