Roassal @ Pharo TechTalk
May 30, 2017

Content of the Roassal TechTalk:
1 - General introduction of Roassal
2 - Latest news in Roassal (Theme support, Scatterplot Matrix, Elastic boxes, ...)
3 - Applications based on Roassal (RTScatterplotMatrix, Test Coverage, Memory profiling)
4 - Ongoing effort and future work

Introduction
Roassal is a visualization engine for Pharo and VisualWorks. Roassal is available under the MIT license.
On Pharo, Roassal may be downloaded from the Pharo catalog,

Alternatively, Roassal can be loaded using the following Gofer expression:

Gofer it
The World menu contains several entries to access the examples, to report bugs, to jump to the agile visualization book (http://AgileVisualization.com), and load addition plugins.

**First example with Roassal**
Open a Playground, and inspect the following expression:

```smalltalk
v := RTView new.
v add: (RTEllipse element).
v
v
```

![Image of World menu](image1)

![Image of Playground](image2)
Many elements can be added:

```smalltalk
v := RTView new.
v addAll: (RTEllipse elementsOn: (1 to: 1000)).
RTGridLayout on: v elements.
v
You can change the size of the circles:
```smalltalk
v := RTView new.
shape := RTEllipse new size: #yourself.
elements := shape elementsOn: (1 to: 100).
v addAll: elements.
elements @ RTPopup.
RTGridLayout on: v elements.
v```
An element is an instance of the class RTElement, and represents a graphical element on the screen. An element points to an object, which could be any objects). A popup indicates which object is represented by an element.

A shape is a factory of elements. Shape may be configured in many different ways. On the example above, size of the element depends on the represented objects.

We have here used a grid layout to order the elements. Without the layout, all the elements will be on the exact same place, which is not really interesting in our case.

Another interesting layout is the flow layout:
```
v := RTView new.
shape := RTEllipse new size: #yourself.
elements := shape elementsOn: (1 to: 100).
v addAll: elements.
elements @ RTPopup.
RTFlowLayout on: v elements.
v
```
At that stage, Roassal appears to be very similar to any graphical libraries. One significant difference is the use of shapes as factories. A shape can be parametrized with metrics and properties that can be applied on the represented object.

Another significant differences are builders. A builder represent a dedicated visualization. The Builder infrastructure is described in a dedicated chapter: https://dl.dropboxusercontent.com/u/31543901/AgileVisualization/Builder/0201-Builder.html

**Mondrian**

Mondrian is a Roassal builder to easily build what is called a *polymetric view*. The example given previous can be written as:

```plaintext
b := RTMondrian new.
b shape ellipse size: #yourself.
b nodes: (1 to: 100).
b layout flow.
b
```
As you can see, the code is slightly shorter using Mondrian. However, Mondrian comes with several large APIs. On the good side, hundreds of examples cover Mondrian.

Edges may be easily defined:

```ruby
b := RTMondrian new.
b shape ellipse size: #yourself.
```

b nodes: (1 to: 100).  
b edges connectFrom: [ :value | value // 2 ].  
b layout tree.

The code given above uses the tree layout.

Here is an example using the cluster layout:

b := RTMondrian new.  
b shape ellipse size: #yourself; color: Color red trans.  
b nodes: (1 to: 100).  
b edges connectFrom: [ :value | value // 2 ].  
b layout cluster.

Normalization is an important aspect of Roassal and Mondrian. Instead of giving an absolute size, we can use a normalizer to set sizes. For example:

b := RTMondrian new.  
b shape ellipse color: Color red trans.  
b nodes: (1 to: 100).  
b edges connectFrom: [ :value | value // 2 ].  
b layout cluster.  
b normalizer normalizeSize: #yourself min: 10 max: 20.
So far, we have used numerical values in our examples. We will visualize files instead:

```plaintext
b := RTMondrian new.
b shape ellipse color: Color red trans.
b nodes: (1 to: 100).
b edges connectFrom: [ :value | value // 2 ].
b layout cluster.
b normalizer normalizeSize: #yourself min: 10 max: 20.
```

```plaintext
b := RTMondrian new.
b shape ellipse color: Color red trans.
b nodes: '/Users/alexandrebergeg/Dropbox/Data/C-Application' asFileReference allChildren.
b edges connectFrom: #parent.
b layout cluster.
b
```
We can represent the size of the files:

```smalltalk
b := RTMondrian new.
b shape ellipse color: Color red trans.
b nodes: '/Users/alexandrebergel/Dropbox/Data/C-Application/SDL/src' asFileReference allChildren.
b edges connectFrom: #parent.
b layout cluster.
b normalizer normalizeSize: #size.
```

Another recurrent example is to visualize classes:

```smalltalk
b := RTMondrian new.
b shape ellipse color: Color red trans; size: #numberOfMethods.
b nodes: RTObject withAllSubclasses.
b edges connectFrom: #superclass.
```
The code above visualize all the classes of Roassal. A tree layout works well in that case:

```
b := RTMondrian new.
b shape box color: Color red trans; height: #numberOfMethods; width: #numberOfVariables.
b nodes: RTOBJECT withAllSubclasses.
b edges connectFrom: #superclass.
b layout tree.
```
Mondrian allow for sophisticated configuration of edges. For example:

```smalltalk
b := RTMondrian new.

b shape circle size: 100;
    color: Color blue trans.

b nodes: (1 to: 2).

b shape arrowedBeziers withOffsetIfMultiple.

b edges useAssociations: { 1->2. 1->2. 1->2. 2->1. 2->1}.
```

---

**Graper**
Grapher is another builder of Roassal to charts.
A very simple example is:

```smalltalk
b := RTGrapher new.\nds := RTData new.
ds points: (1 to: 10).
ds y: #yourself.
b add: ds.
```

```smalltalk
b := RTGrapher new.\nds := RTData new.
ds points: (1 to: 10).
ds y: #yourself.
d ds dotShape color: Color red trans.
ds x: #numberOfLinesOfClass.
ds y: #numberOfMethods.
b add: ds.
```
b := RTGraperh new.
ds := RTData new.
ds dotShape color: Color red trans.
ds points: (Collection withAllSubclasses).
ds x: #numberOfLinesOfCode.
ds y: #numberOfMethods.

b add: ds.
b

Applications

We recently developed RTScatterplotMatrix, a matrix of scatterplots:

b := RTScatterplotMatrix new.
b shape circle color: Color red trans.
b objects: Collection withAllSubclasses.
b metrics
  at: 'LOC' put: #numberOfLinesOfCode;
  at: 'NOM' put: #numberOfMethods;
  at: 'NOV' put: #numberOfVariables.

Hapao is a test code coverage, available under the Spy2 catalog entry. You can visualize the test coverage of the test execution or any execution.
Visualizing the production of objects:

```small
(MSP profile: [ RTMapExample new exampleWorldPopulation ] onPackagesNamed: #( 'Roassal2' 'Trachel' ) ) inspect
```

**Ongoing and Future work**

We are aware of the complexity of the Roassal API. Data Studio is an effort to alleviate this complexity by providing an expressive graphical user interface to build visualizations.