Advanced Rendering of Model Appearance (KARMA)

> ⚠ **Legacy Project**
>
> KARMA is not maintained anymore and hence not part of any KIELER release.

Project Overview

Responsible:

- Miro Spönemann
- Christoph Krüger

Overview

This plugin aims to provide tools that make it easier to change the appearance of the diagram. A diagram using this has to do mainly three things.

- Set up the edit parts to inherit classes provided by this plugin
- Implement a RenderingProvider can describe new Figures that should be displayed as well as a LayoutManager or BorderItemLocator
- Define the conditions under which the Figures should be displayed with an expansion point

EditPart

The following changes that have to be done in the EditParts of all elements that should use the mechanism.

- The EditPart should inherit from a fitting EditPart provided by this plugin. For example an edit part that inherited formerly from ShapeNodeEditPart should now inherit from AdvancedRenderingShapeNodeEditPart.
- The nested Figure should inherit from SwitchableFigure provided by this plugin. This is necessary to preserve the label mechanism that is generated by gmf. Can be skipped if the model element is of a type that does not work on figures i.e.: connections or labels.
- The attribute "primaryShape" in case it exists has to be deleted since it will be provided by inheritance.

These changes are ideally done in the GMF templates so that they can be automatically generated.

RenderingProvider

In the RenderingProvider, implementing the IRenderingProvider given by this project, the figures that should be displayed are defined. The RenderingProvider is given a string written in the extensionpoint, which is used to identify what Figure should be constructed. It could be a path to an svg or just a name to be handled be the developer or anything the user can think of. The RenderingProvider is also given the old figure thats currently displayed. This is for the purpose of just changing a few attributes of the figure instead of building an entirely new one. For Example it is not possible to change the Figure of a connection. It is possible however to change the style of a connection. In this case one would change the style of the given oldfigure and then return that one. Finally it gets the model element. Apart from the figures the RenderingProvider can also describe LayoutManagers and BorderItemLocators in a similar way.

Extension Point

Most work is done by the extension point. Here the RenderingProvider is registered as well as some EPackages that contain features to be used by the conditions, a number of EditParts the mechanism should be applicated to and of course the conditions them self. An extension also has a priority which is used the order of configurations in case there are more than one for the same EditPart. The default priority is 1.

Conditions

FeatureValueCondition

This condition checks for the value of a feature thats part of an ePackage. The first field is the name of the feature itself. The second the name of the classifier that owns the feature. Third the value the feature should have, could be a boolean or an enumeration(feel free to demand support for more if needed). And last the strings that will be given to the RenderingProvider to identify the Figure, LayoutManager and BorderItemLocator to be displayed under this condition.

ListSizeCondition

Similar to feature value condition but instead of a value it has a size. Other than simple numbers the size field also supports things such as >, <, !=, etc.

CompoundCondition

The compound condition works like a logical "and" for all its child conditions.

CustomCondition

The custom condition can be anything that implements ICustomCondition. The key and value fields are given to the initialize method. An example of this is the AnnotationCondition.

AnnotationCondition

This condition tries to get an Annotation from an Annotatable with the name of the key. It then checks for the type of the Annotation and tries to parse the value to an appropriate type and then checks whether it equals the value of the Annotation. This condition works only for direct child annotations. For more hierachy layers use the RecursiveAnnotationCondition which otherwise works in a similar way.

AnnotationExistsCondition

This condition checks if an annotation with the name of the key exists as a child of the given EObject.

Feature Usage Overview

Node

For example states or entities.

Appearance

By setting the figureparam field you can change the appearance of a node. After evaluating the conditions the corresponding string will be given to the rendering provider. Using this string you decide which figure should be drawn in this case. Usually this means having a lot of if else cases und pointing to methods returning an implementations of a draw2d IFigure interface.

Layout

Similar to the appearance you get to decide on the LayoutManager to be used by the figure. This LayoutManager is an implementation of the draw2d LayoutManager interface. The layoutmanager is usually used to determine the size of a node with its children and the positions of child figures.

Size

Its also possible to set a size for the node. The fixed size field accepts values in the form of x,y. Alternatively if no size given here it can also be determined by giving the figure some bounds while doing the appearance or by using the layout manager.

Borderitemlocator

Since the node is likely not a borderitem there is no reason for it to have such a locator, ignore the field in this case.

Borderitems

Such as ports and labels.

Appearance/Layout/Size

can be used the same way a for nodes.

BorderItemLocator

You can give the borderitem a custom borderitemlocator in a similar way you give it a new LayoutManager. Return an Object that implements the gmf IBorderItemLocator interface. A borderitemlocator is used to determine the location of the borderitem relative to its parent object.

Connections

Just that, a connection between two nodelike objects.

Appearance

You can't give a connection a completely new appearance due to technical restrictions. Thats not really a big problem though since connections are usually some kind of line anyway. You can however al least alter the appearance a bit by influencing the old figure given to the rendering provider. For example its possible to change the source and target decorations, having a dashed line, changing the line thickness and so on.

Layout/Size/BorderItemLocator

These fields should be ignored in this case since they are not really fitting to the concept of a connection.