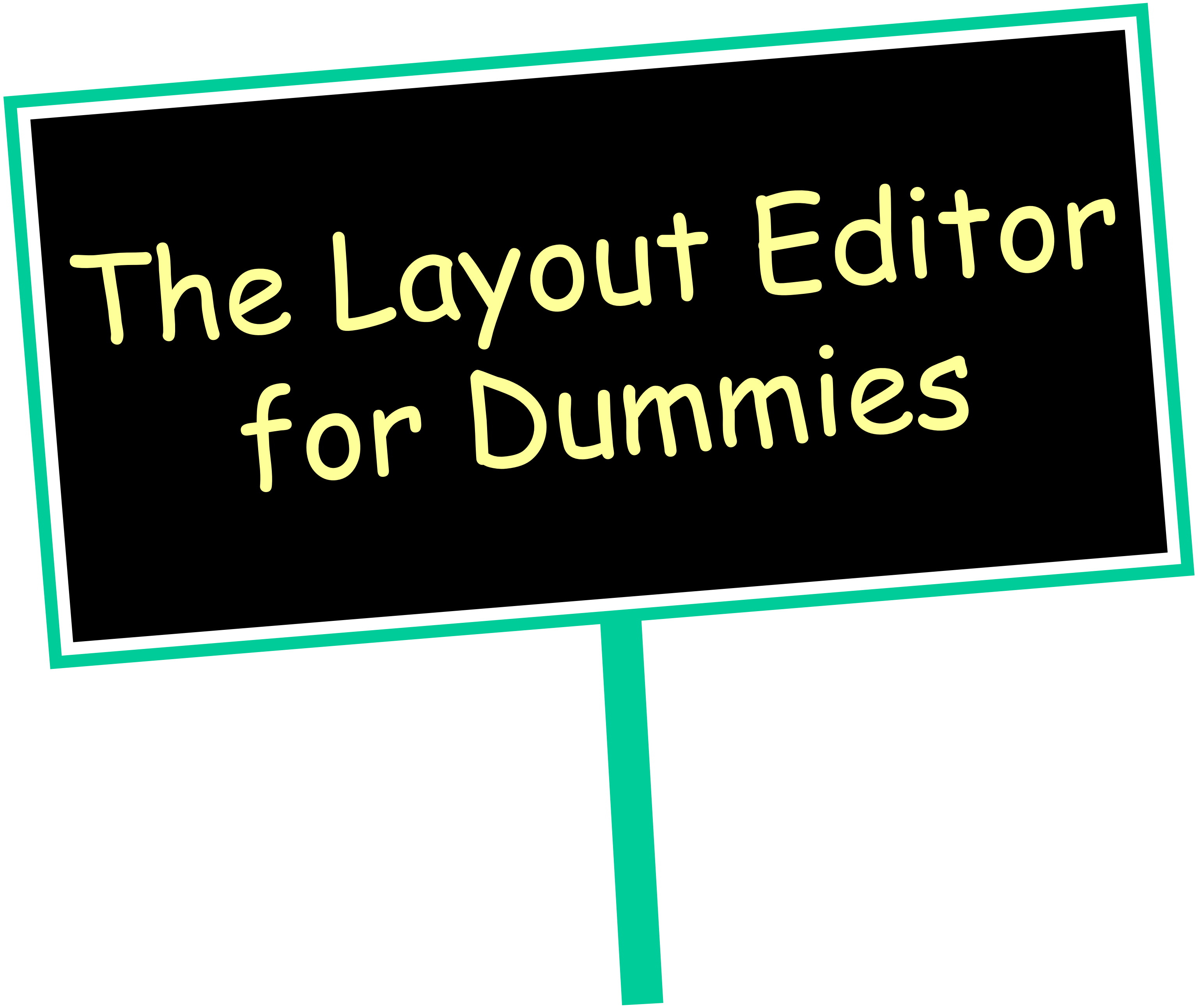

The Layout Editor
for Dummies

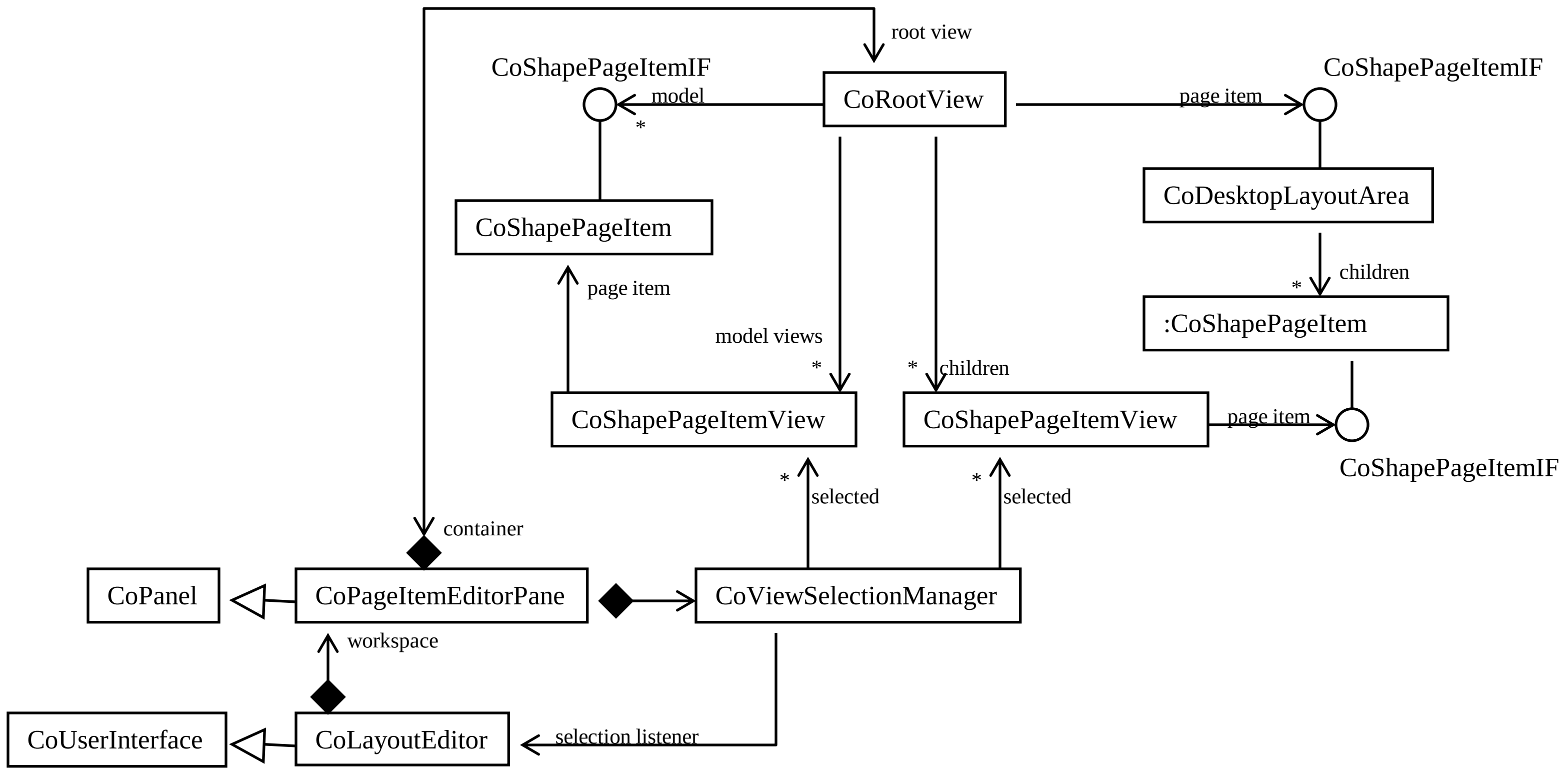
This document

This document contains a set of diagrams that describe the layout editor and its behavior. It is by no means a complete description. Instead it focuses on a few selected aspects of the layout editor. Also, most of the diagrams contains deliberate simplifications (the details are in the code).

The reader is expected to be familiar with the page item model, swing and the BlueBrim UI framework.

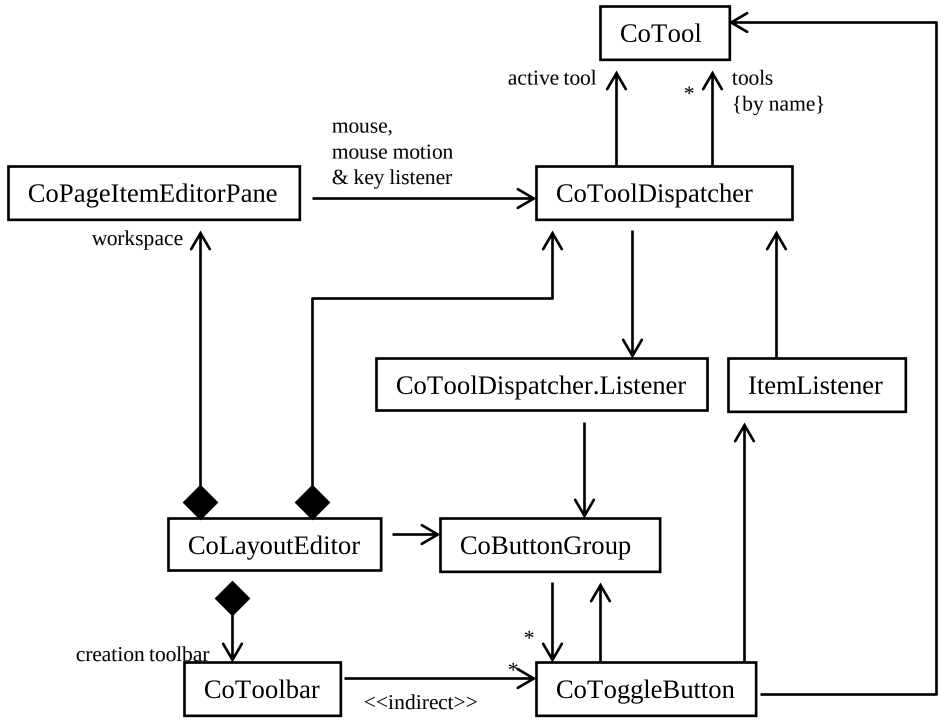
Workspace, root view and selection

This diagram shows how page items are connected to the layout editor.
Included is also the objects involved in page item view selection..



Layout editor tools

Mouse and keyboard behavior in the layout editor is implemented by objects known as tools. The layout editor has a tool dispatcher that manages the tools. The tool dispatcher listens to all mouse and key events in the layout editors workspace and dispatches the events to the tool that is currently active. The tool performs what ever task it is supposed to and returns the tool that is to be the next active one (often the active tool itself).



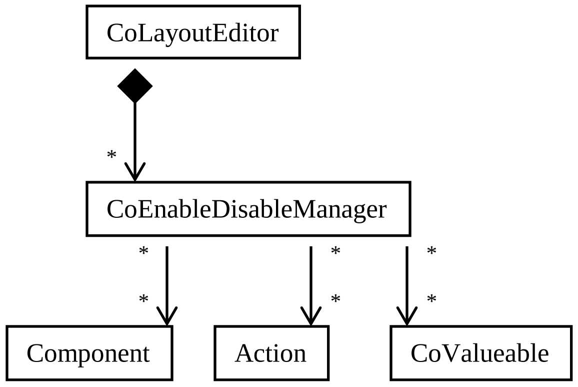
Enable /disable

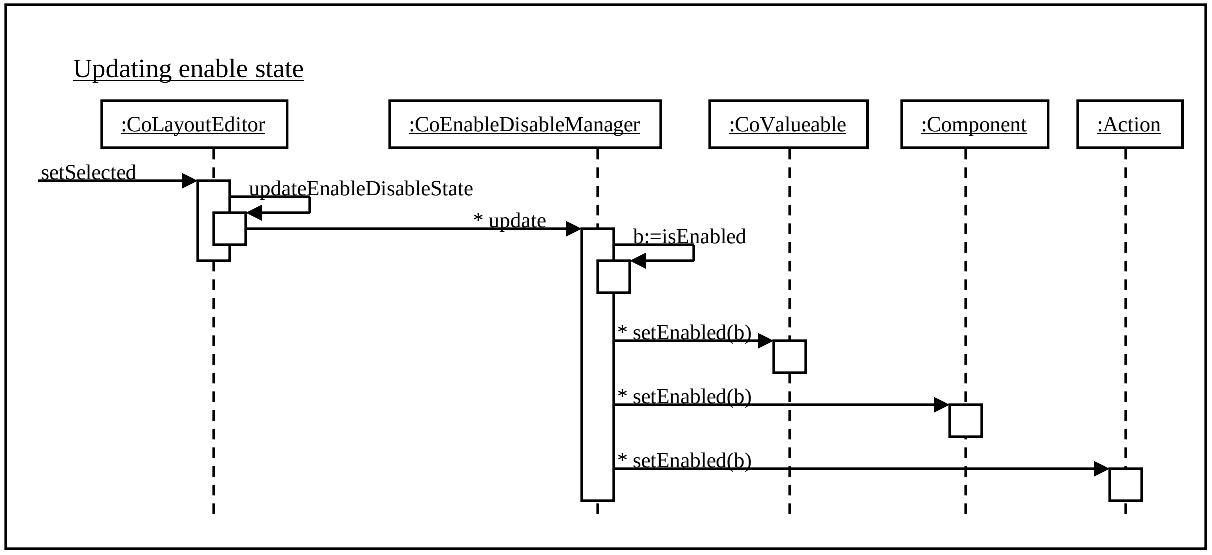
Enabling and disabling of components in the layout editor is handled by a set of CoEnableDisableManagers. Each CoEnableDisableManager represents a particular state. Examples of such states are:

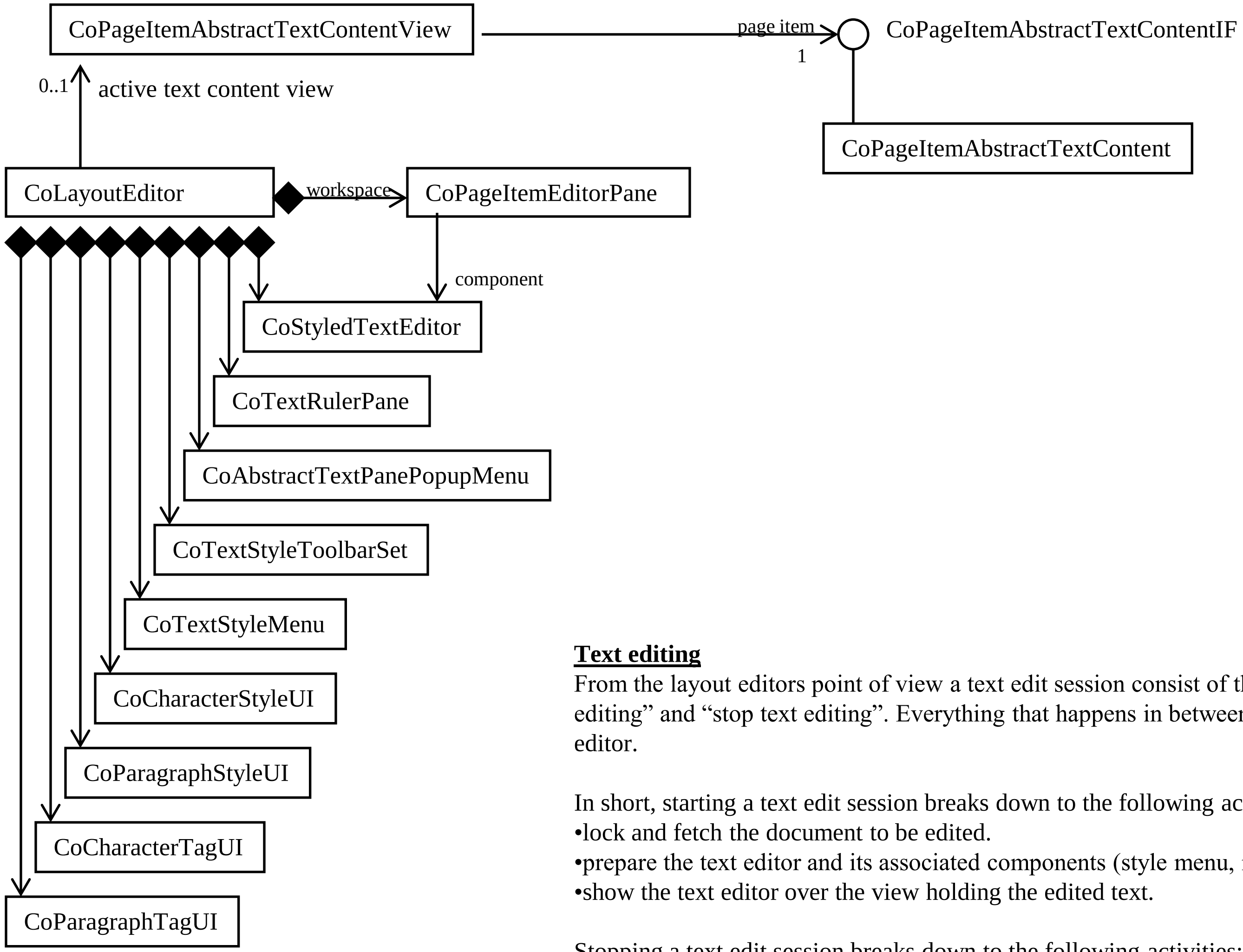
•One or more page items are selected.
•One image box is selected.
•Text editor is active
•...

Associated with every CoEnableDisableManager are the objects whose enable state is the same as that of the CoEnableDisableManager. These objects can be awt Components, Actions or CoValuables.

Whenever the state of the layout editor changes in a way that might effect the enable state of its components then the method updateEnableDisableState() is be called. This causes the CoEnableDisableManagers to set the enable state of their associated objects.







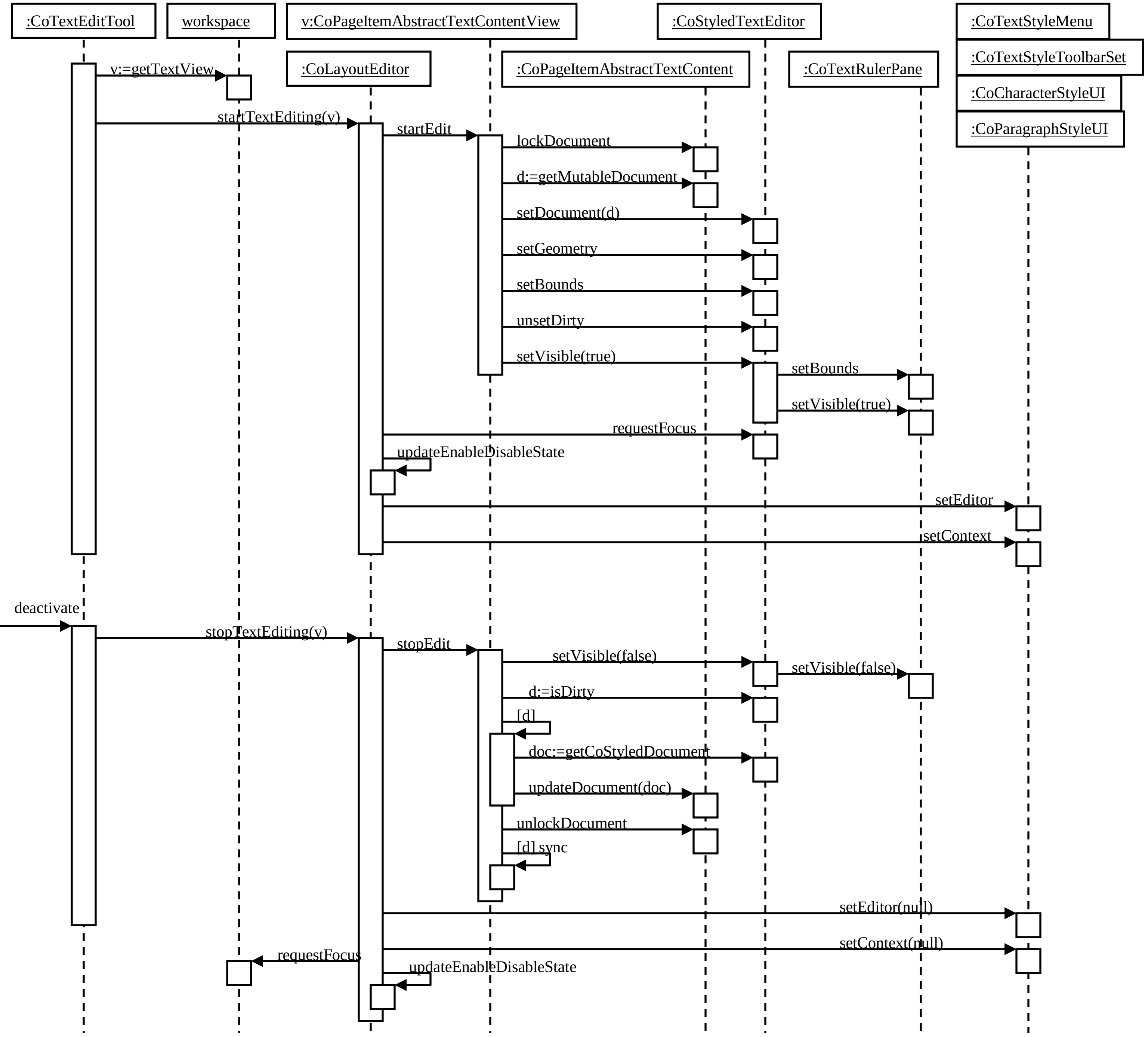
Text editing

From the layout editors point of view a text edit session consist of the two events “start text editing” and “stop text editing”. Everything that happens in between is the responsibility of the text editor.

In short, starting a text edit session breaks down to the following activities:
•lock and fetch the document to be edited.
•prepare the text editor and its associated components (style menu, ruler pane, style ui’s, ...).
•show the text editor over the view holding the edited text.

Stopping a text edit session breaks down to the following activities:
•hide the text editor.
•put the edited document back into the page item from where it came.
•disable the components associated with the text editor.

The sequence diagram on the next page illustrates these two cases in more detail.


:CoTextEditTool
workspace
v:CoPageItemAbstractTextContentView
:CoStyledTextEditor
:CoTextStyleMenu
:CoTextStyleToolbarSet
:CoCharacterStyleUI
:CoParagraphStyleUI
:CoLayoutEditor
:CoPageItemAbstractTextContent
:CoTextRulerPane
v:=getTextView
startTextEditing(v)
startEdit
lockDocument
d:=getMutableDocument
setDocument(d)
setGeometry
setBounds
unsetDirty
setVisible(true)
setBounds
setVisible(true)
requestFocus
updateEnableDisableState
setEditor
setContext
deactivate
stopTextEditing(v)
stopEdit
setVisible(false)
setVisible(false)
d:=isDirty
[d]
doc:=getCoStyledDocument
updateDocument(doc)
unlockDocument
[d] sync
setEditor(null)
setContext(null)
requestFocus
updateEnableDisableState

The modify pane

The modify pane is the primary ui for displaying and manipulating page item state. It contains a set of tabs each containing a page item property panel. The page item property panels contains controls for displaying and manipulating every property of a page item. The type of page item displayed defines which tabs are visible. For instance, the tab displaying image properties is visible only when an image box is the model.

The model of the modify pane is not the page item being displayed. Instead it is the view of that page item. This makes it possible to fetch the page item state without server round trips (caching page item state kind of the point of views anyway). The modify pane register itself as a page item view listener in order to keep its state updated.

When a page item property value is changed the page item reference is fetched from the view and the appropriate manipulating calls are made.

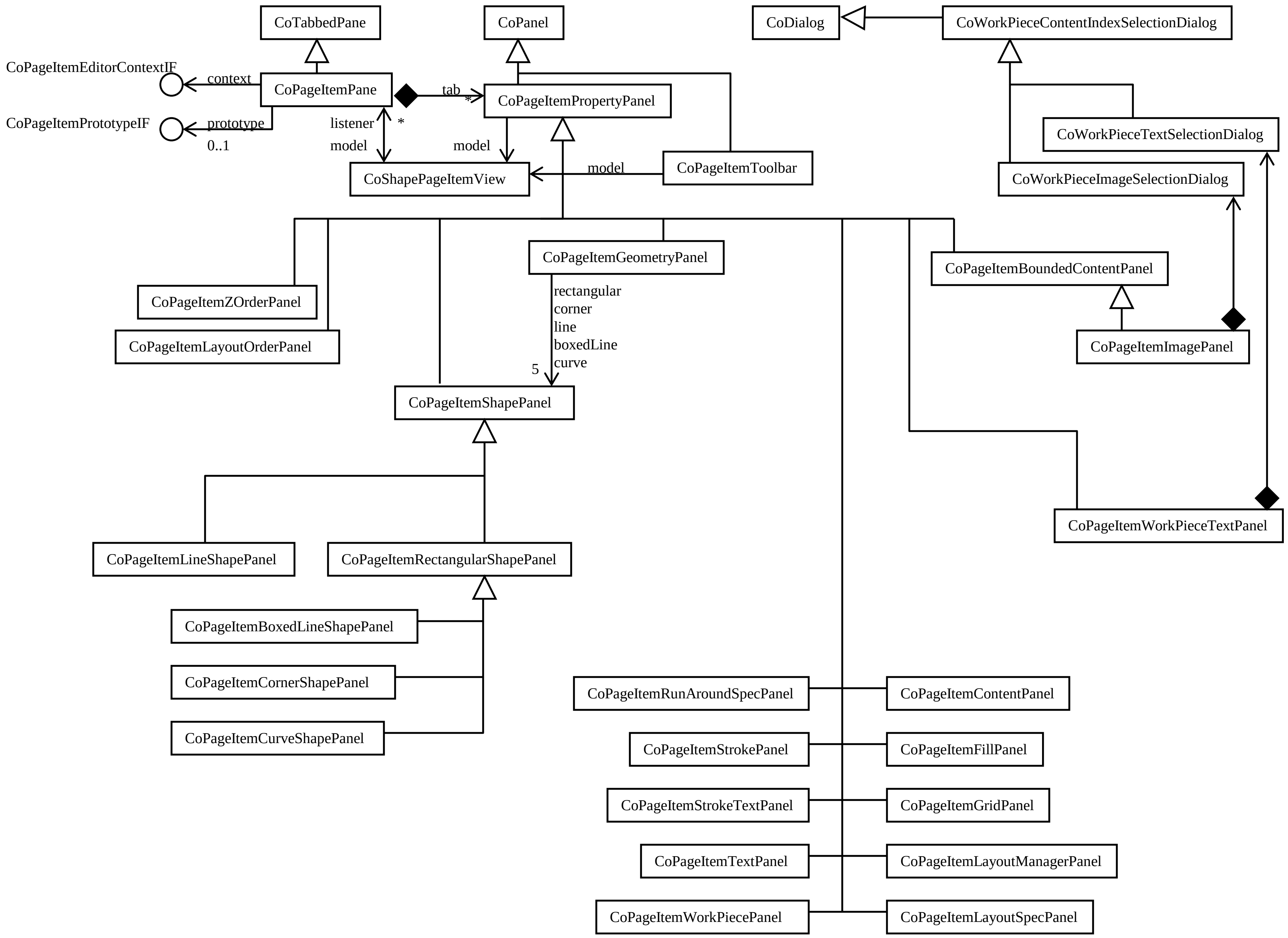
The modify pane can also be used on page item prototypes. Since page item prototypes normally don't have views one must be created before the prototype can be displayed in the modify pane.

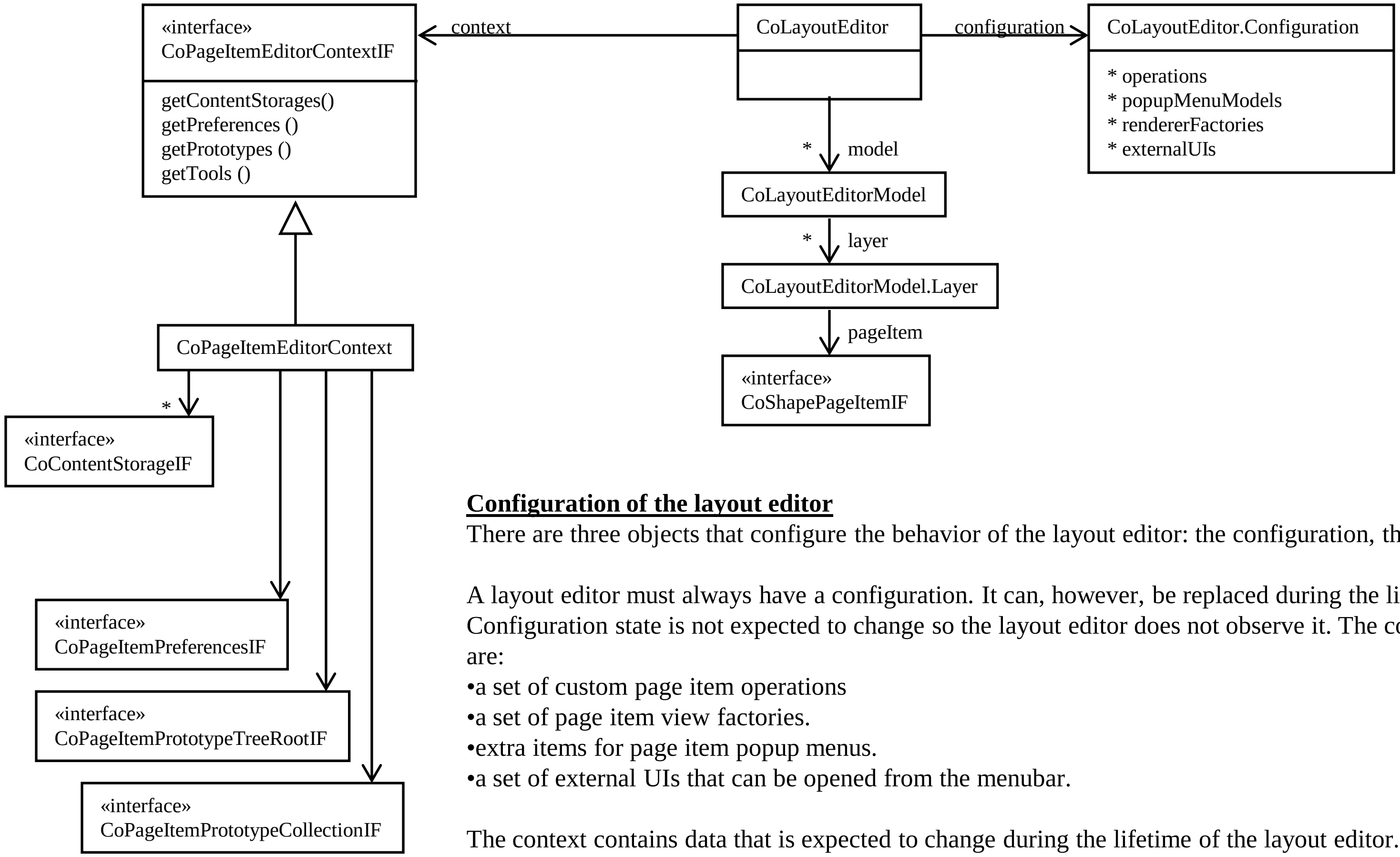
The modify pane needs a context. The context provides information such as what colors are available. The modify pane can live without a context but some of the controls will be useless (selecting the fill color from an empty set of colors isn't very useful).

The page item toolbar

The page item toolbar contains a subset of the controls in the modify pane. Its purpose is to provide a quick (it is always visible, while the modify dialog is displayed in a dialog) yet cheap (screen real-estate wise) way to display and manipulate a selected set of page item properties.

The diagram on the next page shows the static class structure of the modify pane and the page item toolbar.


CoTabbedPane
CoPanel
CoDialog
CoWorkPieceContentIndexSelectionDialog
CoPageItemEditorContextIF
context
CoPageItemPane
tab
*
CoPageItemPropertyPanel
CoPageItemPrototypeIF
prototype
0..1
listener
*
model
model
CoWorkPieceTextSelectionDialog
CoShapePageItemView
model
CoPageItemToolbar
CoWorkPieceImageSelectionDialog
CoPageItemGeometryPanel
CoPageItemBoundedContentPanel
rectangular
corner
line
boxedLine
curve
5
CoPageItemZOrderPanel
CoPageItemLayoutOrderPanel
CoPageItemImagePanel
CoPageItemShapePanel
CoPageItemWorkPieceTextPanel
CoPageItemLineShapePanel
CoPageItemRectangularShapePanel
CoPageItemBoxedLineShapePanel
CoPageItemCornerShapePanel
CoPageItemCurveShapePanel
CoPageItemRunAroundSpecPanel
CoPageItemContentPanel
CoPageItemStrokePanel
CoPageItemFillPanel
CoPageItemStrokeTextPanel
CoPageItemGridPanel
CoPageItemTextPanel
CoPageItemLayoutManagerPanel
CoPageItemWorkPiecePanel
CoPageItemLayoutSpecPanel



Configuration of the layout editor

There are three objects that configure the behavior of the layout editor: the configuration, the context and a set of models.

A layout editor must always have a configuration. It can, however, be replaced during the lifespan of a layout editor. Configuration state is not expected to change so the layout editor does not observe it. The contents of a configuration object are:
•a set of custom page item operations
•a set of page item view factories.
•extra items for page item popup menus.
•a set of external UIs that can be opened from the menubar.

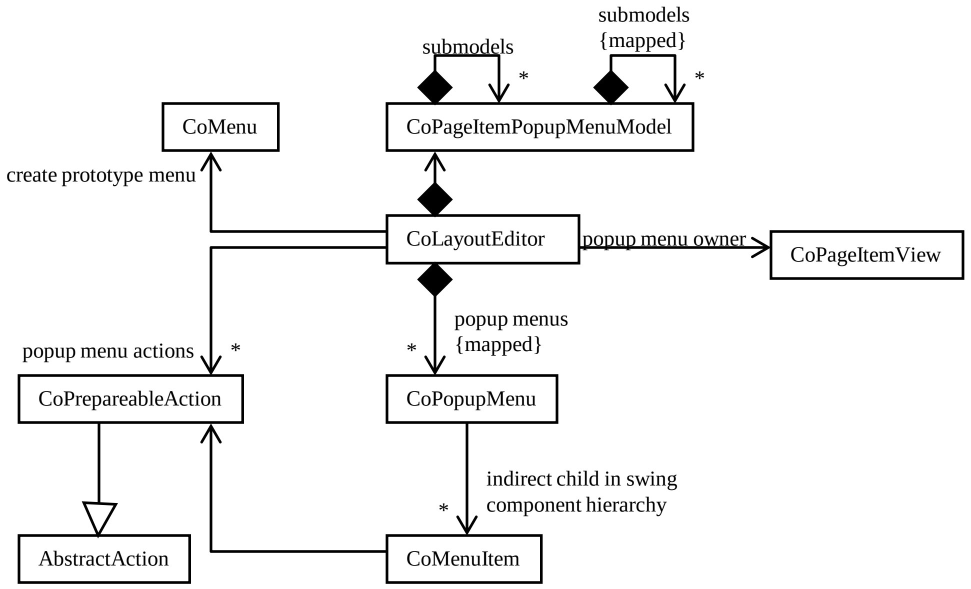
The context contains data that is expected to change during the lifetime of the layout editor. Therefore it is observed by the layout editor. Replacing the context with another context triggers the same update process as a notification of context change. The contents of a context object are:
•a CoPageItemPrefrences object containing available colors, strokes and such.
•a tree structure of page item templates.
•a set of page item prototypes from which the creation tools are derived.
•a set content storages (used when creating content from the layout editor).

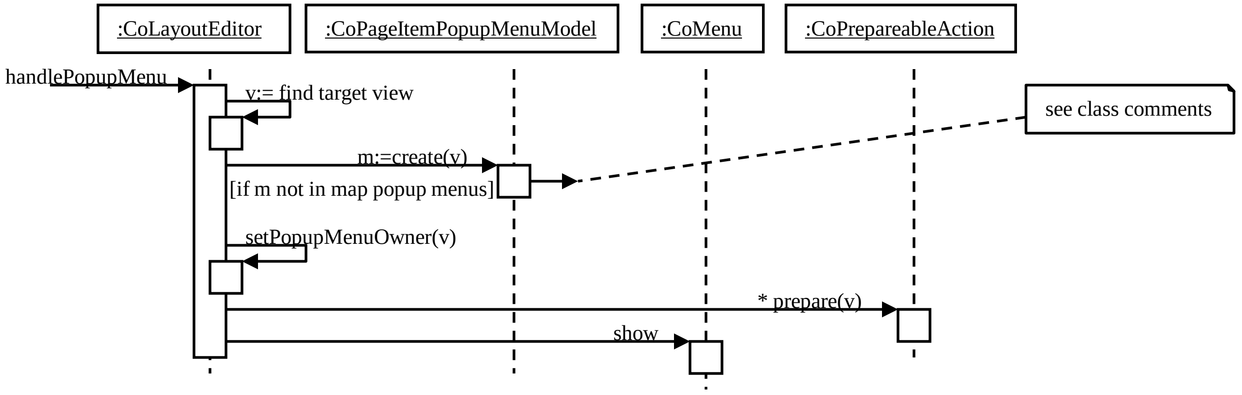
The models of the layout editor is the collection of page items that are edited. Naturally they are expected to change (that is what the layout editor does). If multiple models are supplied they are laid out in a left to right fashion. Each model consists of a set of layers of which one is active at any given time. Each layer corresponds to a page item. Only page items that are active can be edited.

Popup menus

Some page item operations are performed using popup menus. Since different page item "types" have different sets of applicable operations, each page item "type" type must have a unique popup menu. This achieved by keeping a map from popup menu keys to popup menus. The popup menu keys are supplied by the page item views.

The popup menus are created by a popup menu model. For details on popup menu models, see the comments in the code.





Page item drag

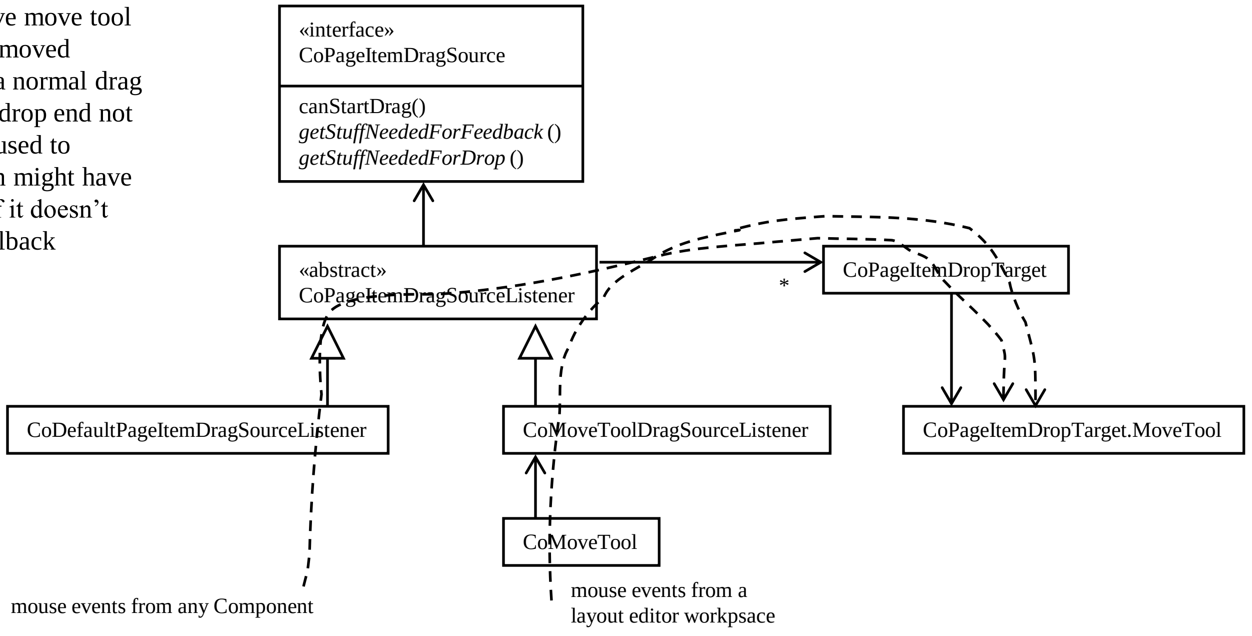
Page item drag can start in two different ways. A swing component can be prepared for page item drag by creating a CoDefaultPageItemDragSourceListener. This is used in the page item holder palette ui. The other way is to move a page item outside the workspace component in a layout editor. When this happens the active move tool will create a CoMoveToolDragSourceListener and dispatch the mouse events to it instead of handling them itself. In both cases the mouse events drive the dnd engine in the class CoPageItemDragSourceListener. This class finds the drop target (if any) and forwards the mouse events to it.

Page item drop

Currently the only one page item drop target is the layout editor. When a drag operation enters the workspace of a layout editor a temporary move tool is created and mouse events are forwarded to it, giving the impression of a normal page item move. If a drag operation enters the layout edior workspace it originated from the dnd session is canceled and the original move tool starts handling mouse events in a normal fashion.

Future work

The only reason page items have their own DnD mechanism is the fact that in order to provide proper dropsite feedback the dragged data (page item or view) must be available during the drag. In the standard DnD mechanism the dragged data is mn't ade available until the drop is performed. However this can probably be worked around by passing this data as part of the DataFlavor object. Since page item views are serializable it might even work across VMs but I wouldn't bet on it. Anothe issue to be adressed is the way an active move tool temporarily hands control to the DnD engine when it is moved outside the layout editor workspace. Some way to start a normal drag session when his happens must be implemented. At the drop end not many changes are needed. As now, a move tool can be used to implement feedback. However, the drop implementation might have to handle drops where no page item view is available (if it doesn't work across VMs). In this case some kind of simple fallback feedback behavior is needed.


«interface»
CoPageItemDragSource
canStartDrag()
getStuffNeededForFeedback ()
getStuffNeededForDrop ()
«abstract»
CoPageItemDragSourceListener
CoPageItemDropTarget
*
CoDefaultPageItemDragSourceListener
CoMoveToolDragSourceListener
CoPageItemDropTarget.MoveTool
CoMoveTool
mouse events from any Component
mouse events from a
layout editor workpsace