Tablet Subsystem

This subsystem constitutes the entire Drawing Domain. It provides a layer of abstraction between the Flatland Model Diagram Layout Application domain and the graphics library.

Relationship numbering range: R1-49

Class Descriptions

Asset

Any given Drawing Type specifies the kinds of text or graphical Elements that might be drawn on it. Each kind of Element is defined as an Asset. A `class name` might be an Asset defined for the drawing type `Executable UML Class Diagram` fore example. A `solid dot` might be an Asset defined for an `Executable UML State Machine Diagram`. The Asset abstraction makes it possible to define the various presentation characteristics such as font size or color fill that can be used to render any associated Elements defined by that Asset. Thus, all `class names` with the same Presentation will be rendered with a common style.

Identifiers

1. **ID**

Singleton value

Attributes

ID

This is a singleton, so any value is fine.

Type: Nominal

Presentation

Same as **Presentation.Name**

Drawing type

Same as **Drawing Type.Name**

Size

The extent of a rectangular drawing area expressed in points.

Type: Rect Size

Asset Presentation

To define the styles that are applied when rendering an Asset, a Presentation is applied. See relationship description for details.

Identifiers

1. **Asset** + **Presentation** + **Drawing type**

Attributes

No non referential attributes

Closed Shape

This is a Shape Element whose line segments form a polygon. As such a fill must be specified in addition to any border style.

Identifiers

1. **ID** + **Layer**

Attributes

No non-referential attributes

Color Usage

By default, a closed shape may be filled in with a color based on the shape's definition as an Asset within a Presentation. An Asset representing an xUML class compartment in a default Presentation, for example, might get a white fill.

But the user may designate some special usage for that shape and pass along a usage name. This usage name will be mapped to some predefined color. So whenever a shape asset is marked for a Color Usage, the associated color is looked up and used as the shape fill instead of any Closed Shape Fill that would normally have been applied.

Identifiers

1. Name

Attributes

Name

A name indicating usage of a shape asset such as 'error', 'normal', 'specification', 'timer' etc.

Type: Color Usage Name based on the Name type

Drawing Type

A mix of graphic and text content commonly used together to fashion a particular type of drawing is regarded as a Drawing Type. An example might be a certain kind of model diagram, say an `Executable UML Class Diagram'`. Drawing Types are not limited to model diagrams, however. A frame of headers, footers and a title block for a certain range of sheet sizes is also a Drawing Type. A set of annotations scattered across a sheet with callout lines might also be a Drawing Type.

Multiple Drawing Types might be layered on a Tablet. This makes it possible to create a model diagram with one set of styles (Presentation) combined with a header-footer frame using its own Presentation. And that might even be combined with a layer of annotations using yet another Presentation.

Identifiers

1. Name

Attributes

Name

A name indicating the purpose and size (if there is more than one) of a drawing's style requirements. Examples could be `xUML class diagram` or `xUML class diagram large` or `OS Engineer Frame D` and so on.

Type: Drawing Type Name based on Name system type

Element

A unit of text or graphics placed and rendered on a Layer and styled by an Asset.

Identifiers

1. **ID** + **Layer**

Attributes

ID

A unique identifier

Type: Element ID based on the Nominal system data type

Size

The extent of a rectangular drawing area expressed in points.

Type: Rect Size

Layer

A common feature of many drawing and image editing applications is the ability to stack layers of content along a z axis toward the user's viewpoint. Similarly, the Tablet renders each layer working from the lowest value on the z axis toward the highest. Thus, content on the higher layers may overlap content underneath.

Identifiers

1. Name
2. Z coord

Since there is only one Tablet instance, we can don't need to include the Tablet referential attribute in either identifier.

Attributes

Name

Describes the overall purpose or reason why content should be drawn together at the same level. Examples might be "diagram", "notes", "header footer", etc.

Z coord

This value indicates the z axis ordering with lower numbered values being drawn underneath higher valued numbers. So Level 1 is drawn first working upward. (To be user, rather than programmer friendly, our Tablet z-axis starts at 1 going upward).

Type: Ordinal

Line Segment

This is a Shape Element that connects two points with a straight line.

Identifiers

1. **ID** + **Layer**

Attributes

From

When drawing a Line Segment it is necessary to draw from one position to another on the Tablet. In fact, it makes no difference which direction you draw, so the from/to naming could just as easily be named a/b. But many draw facilities use this terminology, so we adopt it here.

Type: Tablet Coord

To

See **From** attribute description

Type: Tablet Coord

Polygon

A series of line segments that wrap around to form a closed shape.

Identifiers

1. **ID** + **Layer**

Attributes

No non-referential attributes

Presentation

A set of compatible visual styles including fonts, colors, border widths and so forth as appropriate to a given Drawing Type form a selectable Presentation. For example, an `Executable UML State Machine Diagram` might be drawn using certain fonts for state names and possibly different colors for transient and non-transient states. Alternatively, only black and white might be used with purple for a certain kind of connector in a diagnostic Presentation.

Identifiers

1. Name

Attributes

Name

A name describing the overall appearance and purpose of the Presentation such as 'diagnostic connectors' or 'normal' or 'color coded states'.

Type: Presentation Name based on Name system type

Rectangle

A Rectangle is specified with a lower left corner and width/height size. While it is technically a polygon, most drawing facilities provide a separate method for rendering rectangular shapes.

Identifiers

1. **ID** + **Layer**

Attributes

Size

The height and width of the Rectangle on the Tablet.

Type: Rect Size

Lower left

The position of the lower left corner in Tablet coordinates.

Type: Tablet Coord

Shape Element

This is an Element rendered directly using line segments or closed shapes with border widths, fill colors and so forth.

Identifiers

1. **ID** + **Layer**

Attributes

No non referential attributes.

Tablet

A virtual drawing surface serving as a proxy for the drawing area provided by the graphics library.

Identifiers

1. **ID**

Attributes

ID

This is a singleton, so any value is fine.

Type: Singleton

Size

The extent of a rectangular drawing area expressed in points.

Type: Rect Size

Text Element

This is an Element rendered using text facilities and styles rather than directly as line segments.

Identifiers

1. **ID** + **Layer**

Attributes

Content

The text to be rendered

Type: Text

Lower left

The lower left corner of the text area.

Type: Tablet Coord

Usage Fill

For any given usage we predefine a color mapping. For example, we could define the 'error' Color Usage to be associated with 'bubble gum' (pinkish-red) color. Now whenever a Shape Presentation is marked as 'error' it will be filled with that color.

Identifiers

1. **Shape asset** + **Presentation** + **Drawing type**

Based on association multiplicity. There can be at most one Usage Fill designated for a Shape Presentation. See R25 for more details.

Attributes

No non-referential attributes

Vertex

A point in a Polygon where two line segments meet to form a corner.

Identifiers

1. **Polygon** + **Layer** + **Position**

Note that no two vertices may overlap on the same Layer. This 'safety' feature prevents undesirable drawing artifact.

Attributes

Position

The location of the Vertex on the Layer

Type: Tablet Coord

Relationship Descriptions

OR20 / Ordinal

Layer is rendered above

Layers are rendered in ascending Z coord order. Since each Layer is at its own dedicated Z coord and all Layer instances are on the same (only) Tablet instance, there is a total ordering on all Layer instances.

Formalization

Layer.Z coord

R1 / 1:M

Drawing Type appears as defined by *many* **Presentation**

Presentation defines appearance of *one* **Drawing Type**

We initially define a single Presentation for each Drawing Type. For an `Executable UML Class Diagram`, for example, we might use only black for fonts and borders, choose a basic font with one size for class names and another for attributes, methods, relationship names and so forth: easy to read, but nothing special. Later, we might decide that it would be nice to use dashed lines and light green for imported class node types. Eventually, there might be a choice of several Presentations that could be selected to display an `Executable UML Class Diagram`.

A Presentation is specific to a given Diagram Type since it applies styles to particular Node and Connector Types relevant to its Diagram Type.

Formalization

Presentation.Drawing type -> Drawing type.Name

R2 / 1:Mc

Presentation styles content of *zero, one or many* **Layer**

Layer content is styled by *one* **Presentation**

When a Layer is defined, it must be associated with a Presentation. Then, when a given Asset is drawn on that Layer it is known what graphic and text styles to apply. A Layer that draws a title block, for example, will apply an appropriate Presentation for that while another Layer for a model diagram would use a different Presentation.

It is also possible to split a diagram to multiple layers so that the content of one part of the diagram might be styled differently than content on another part. For example, you might want separate Presentations (colors perhaps) for adjoining subsystems.

Formalization

Layer.Presentation -> Presentation.Name

Layer.Drawing type -> Presentation.Drawing type

R4 / M:M-1

Presentation stylizes *one or many* **Asset**

Asset is styled by *one or many* **Presentation**

To draw an Element on a Layer, it is necessary to use the set of styles defined for the Element's Asset. So, let's say that you want to draw a `class compartment` Asset on an `xUML class diagram` Drawing Type using the `default` Presentation. You simply specify the Asset and Layer. The Presentation defined for that Layer will be associated with your Asset and then provide a set of styles (line width, line style, fill color) necessary to draw your Asset (a filled rectangle in this case).

By definition, a Presentation defines a set of styles to be applied to all the Assets defined for the Presentation's Drawing Type. Since a Drawing Type must included at least one Asset, there must be at least one in every Presentation.

A given Asset cannot be drawn unless it has the relevant styles defined. Since a Drawing Type must specify at least one Presentation, there must be at least one Presentation for each Asset.

Formalization

Asset Presentation.Asset -> Asset.Name

Asset Presentation.Presentation -> Presentation.Name

Drawing type -> Presentation.Drawing type

Drawing type -> Asset.Drawing type

Constraint: Note that both the Asset and Presentation must be defined for the same Drawing type, formalized by the merged Drawing type referential attribute.

R7 / 1:Mc

Text Style sets text drawing characteristics of *zero or many* **Text Presentation**

Text Presentation has text drawing characteristics defined by *one* **Text Style**

We want to be able to render a text asset, let's say `class name` in an `xUML class diagram` Diagram Type with a `default` Presentation, with a certain font, size and so forth.

But we don't want the user to have to specify all of these properties every time. Instead, the user will simply specify the Asset combined with its Presentation (Text Presentation) and that alone will be sufficient to identify all of the predefined drawing characteristics.

The same Text Style can be useful for drawing many Assets. `state name` and `class name` for example may use the same text properties.

Formalization

Text Presentation.Text style -> Text Style.Name

R12 / Generalization

Shape Element is a **Line Segment**or **Closed Shape**

Shape Elements all required line style specifications. Closed Shapes additionally require a fill specification.

Formalization

<subclass>.ID -> Shape Element.ID

<subclass>.Layer -> Shape Element.Layer

R13 / 1:M

Tablet organizes content on z axis with *one or many* **Layer**

Layer holds z axis coordinate content of *one* **Tablet**

A Layer is a horizontal slice of content at some position on the z axis. To draw an Element on a Tablet, the Element must be placed on some Layer. To be useful then, a Tablet must be created with an initial Layer. More can be added later.

Since there is only one Tablet instance, all Layers present content for that Tablet.

Formalization

Layer.Tablet -> Tablet.ID

R15 / Generalization

Element is a **Shape**or **Text Line**

Each Element is drawn somewhere on a single Layer. Effectively, then, an Element has an x,y,z position with z being the Layer and x,y being a position on that layer. The x,y might be a lower left corner or an end point depending on whether the Shape is open or closed.

Since the applicable styles differ for Shapes (open and closed) and lines of text (Text Lines), Elements are classified accordingly.

Formalization

<subclass>.ID -> Element.ID

R19 / 1:Mc

Element is drawn on *one* **Layer**

Layer draws *zero, one or many* **Element**

To organize and layer all graphical content, each Element is placed on a specific Layer when it is created.

A blank Layer isn't much use since it won't render any content. That said, it may be necessary to create one without knowing in advance whether or not any content will be placed on it. Since blank Layers are harmless, there is no reason to preclude them.

Formalization

Element.Layer -> Layer.Name

R22 / Generalization

Closed Shape is a **Rectangle**or **Polygon**

Here we distinguish based on how the rendering of a shape is specified. In the case of a simple Rectangle there is generally a direct facility for drawing one using a corner and size. But in the case of a Polygon, especially one that is irregular, a set of Vertex positions must be specified.

Both types of Closed Shape require a fill specification.

Formalization

<subclass>.ID -> Closed Shape.ID

<subclass>.Layer -> Closed Shape.Layer

R23 / 1c:Mc

Shape Presentation applies *zero or one* **Color Usage**

Color Usage is applied to *zero, one or many* **Shape Presentation**

If no Color Usage is supplied for a given Shape Presentation, the shape will be filled with its designated Closed Shape Fill, of any, otherwise, no fill will be applied.

Since only one fill color may be displayed in a closed shape on a layer, at most one can be specified.

The same usage can be applied to a variety of Shape Presentations. For example a Color Usage named 'specification' could be useful both for a specification class in an xUML, Shlaer-Mellor or Starr class diagram. More widely, a Color Usage such as 'imported' could be used in class, collaboration or any other diagram that imports external elements.

And you can certainly define a Color Usage that is anticipated for future use that doesn't happen to be associated with any Shape Presentation yet.

Formalization

Usage Fill.Shape asset -> Shape Presentation.Asset

Usage Fill.Presentation -> Shape Presentation.Presentation

Usage Fill.Drawing type -> Shape Presentation.Drawing type

R24 / 1:Mc

RGB Color represents *zero, one or many* **Color Usage**

Color Usage is represented by *one* **RGB Color**

A Color Usage is essentially a multi-purpose alias for an RGB Color. But it is important that a given usage map to a particular color to yield the desired drawing.

A given RGB Color might be used multiple times. Color Usages 'imported' and 'external', for example, might both map to the color 'blue steel'.

Formalization

Color Usage.Color -> RGB Color.Name

