Node Subsystem

This subsystem considers the Canvas and Diagram as a whole, the grid system for Node placement, the Notation applied to the Diagram and the various types of Nodes that may be placed on the Diagram. Connectors are modeled in a different subsystem.

Relationship numbering range: R1-R49

Class Descriptions

Diagram Layout Specification

Defines a set of values that determine how a Diagram and Grid is positioned on a Canvas and how Nodes are positioned relative to the Diagram and Grid.

Attributes

Name

In this version there is assumed to be only a single specification instance, so the name is here merely expresses unique model identity.

Type: Name

Default margin

The distance from each canvas edge that may not be occupied by the Diagram.

Type: Padding

Default diagram origin

The lower left corner of the Diagram in Canvas coordinates.

Type: Position

Default cell padding

The distance from each Cell edge inward that may not be occupied by any Node. This prevents two Nodes in adjacent Cells from being too close together.

Type: Padding

Default cell alignment

The horizontal and vertical alignment of a Node in its Cell or Cells

Type: Padding

Identifiers

Name

Diagram Notation

A Notation supported by the Flatland draw engine to render Diagrams of a given Diagram Type. See R32 for more details.

Attributes

Diagram type

Same as **Diagram Type.Name**

Notation

Same as **Notation.Name**

Identifiers

Diagram type + **Notation**

Consequence of a many-many association

Diagram Type

A standard diagram such as ‘class diagram’, ‘state machine diagram’ or ‘collaboration diagram’. Each of these types draws certain kinds of Nodes and Connectors supported by one or more standard Notations.

Attributes

Name

A commonly recognized name such as ‘class diagram’, ‘state machine diagram’, etc.

Type: Name

Identifiers

Name

Node

On Diagrams, model entity semantics such as states, classes, subsystems and so forth can be symbolically represented as polygonal or rounded shapes. These shapes can then be connected together with lines representing model relationship semantics. A Node represents the placement of a shape symbol at a specific location (Cell) on a Diagram.

Every Node, regardless of its specific shape as determined by its Node Type, is considered to be roughly or completely rectangular. This means that every Node has four faces, top, bottom, left and right where one or more Connectors may attach.

Attributes

ID

Each Node is numbered uniquely on its Diagram.

Type: Nominal

Node type

Type: Same as **Node Type.Name**

Diagram type

Type: Same as **Node Type.Diagram type**

Size (derived)

The height and width of the Node. This height is derived from the combined heights of its visible Compartments. The width is determined as a result of computing the required width of all of the visible Compartments.

Type: Rect Size

Location

The lower left corner of the Node relative to the Diagram.

Type: Diagram Coordinates

Identifiers

ID

We only handle one Diagram at a time, so the Node.ID is always unique.

Node Type

Specifies characteristics common to all Nodes of a given type. A class node, for example, has three compartments, sharp corners a certain border style, etc. For now, to support a different visual style for a class node, let's say, you would need to define a new node/diagram type combination (UML class on a UML class diagram type vs. Shlaer-Mellor class on a Shlaer-Mellor class diagram type), for example). Since, most diagrams we are considering have notational variation in the Connector Types and not the Node Types, we're baking in the visual characteristics of a Node Type for now and making it flexible for Connector Types.

Attributes

Name

A name like "class", "state", "imported class", "domain", etc.

Type: Name

Diagram type

Type: Same as **Diagram Type.Name**

Rounded

Whether or not all four node corners are rounded

Type: Boolean

Compartments

The number of UML style text compartments visible.

Type: Count1 `:: integer > 0`

Border

Type: Border style

Default size

Initial assumption about a Node size.

Type: Rect Size

Max Size

Node may not be drawn larger than this size.

Type: Rect Size

Corner margin

The minimum distance permitted between a Stem Root end and the nearest Node corner. The intention it to prevent lines attaching on or very close to a Node's corner which looks glitchy.

Type: Distance

Identifiers

Name + **Diagram type**

Notation

A standard (supported by a large or small community) set of symbols used for drawing a Diagram Type.

Attributes

Name

A name such as ‘xUML’, ‘UML’, ‘Starr’, ‘Shlaer-Mellor’, etc.

Type: Name

Identifiers

Name

Relationship Descriptions

R30 / 1:1c

Diagram is rendered using *one* **Diagram Notation**

Diagram Notation renders *zero or one* **Diagram**

When a Diagram is created, there may be a choice of multiple Notations that it can be displayed in. A class diagram, for example, could be displayed as Starr, xUML or Shlaer-Mellor notation. Each potential Diagram would mean the same thing, but the drawn notation would be different in each case.

A Diagram can use only a Notation that is defined for its Diagram Type. Since a Diagram Type must be supported by at least one Notation, there will always be at least one possible choice.

Only one Diagram is rendered at a time. This means that while, in theory, the same Diagram Notation could render multiple Diagrams and certainly does over time, during the runtime of the draw engine, a given Diagram Notation either is or isn't the one that determines the look of a Diagram, thus the 1c multiplicity in this association.

Formalization

Diagram.Notation -> Diagram.Notation.Notation and Diagram.Type -> Diagram Notation.Notation and Diagram Type.Name

The shared Diagram.Type value enforces the constraint that a Diagram's notation must be supported by its specified Diagram Type on R11.

R7 / 1:Mc

Compartment is filled by *zero one or many* **Text Line**

Text Line fills *one* **Compartment**

.

Formalization

Reference in Text Line class.

R8 / 1:Mc

Compartment is a **Title** or **Data Compartment**

.

Formalization

Subclass references to superclass.

R1 / Ordinal

Compartment Type is stacked above

Vertical stacking of corresponding Compartment Types of a Node Type. For example a title compartment is drawn above an attribute compartment which is drawn above a method compartment in a class diagram.

Formalization

Compartment Type identifier I2 with **Stack order**, numbered within **Node type** and **Diagram type**

R32 / M:Mc-1

Diagram Type is supported by *one or many* **Notation**

Notation supports *zero, one or many* **Diagram Type**

The term 'supports' should not be confused with 'compatible'.

Compatibility means that a Notation has been defined, in the real world, to be used with a certain kind of diagram. Support means that the Flatland draw engine currently has the ability to draw a particular Diagram Type in a specified Notation.

Here we assume that compatibility is understood when this relationship is populated and that a given Notation is associated only with those Diagram Types where it makes sense to use it.

For example, the xUML notation is relevant to a wide variety of diagram types, but for now it may only be supported for class diagrams and state machine diagrams. On the other hand, the Starr notation applies only to class diagrams.

So this relationship represents which Notations have been selected to support certain Diagram Types supported by the Flatland drawing tool.

So that they can be drawn, it is essential to ensure that at least one compatible Notation is supported for each Diagram Type defined in the Flatland draw engine.

Formalization

Diagram Notation association class

R11 / 1:1c

Diagram Type specifies *zero or one* **Diagram**

Diagram is specified by *exactly one* **Diagram Type**

A Diagram Type embodies a diagramming standard and so constrains a Diagram to be drawn a certain way, with certain types of Nodes and Connectors. The associated Notation further constrains the drawn look of these elements.

A Connector Type, say a binary association which has meaning in a class diagram won't be available in a state machine diagram, for example.

Therefore, a Diagram is always specified by a single Diagram Type. A given Diagram Type may, or may not be the Diagram Type employed to constrain the currently managed Diagram.

Formalization

Diagram.Type

